Analyses and Validation of Conditional Dependencies with Built-in Predicates

Analyses and Validation of Conditional Dependencies with Built-in Predicates

Wenguang Chen[1], Wenfei Fan[2,3], and Shuai Ma[2]

[1] Peking University, China
[2] University of Edinburgh, UK
[3] Bell Laboratories, USA

Abstract. This paper proposes a natural extension of conditional functional dependencies (CFDs [14]) and conditional inclusion dependencies (CINDs [8]), denoted by CFD^{p}s and CIND^{p}s, respectively, by specifying patterns of data values with $\neq, <, \leq, >$ and $\geq$ predicates. As data quality rules, CFD^{p}s and CIND^{p}s are able to capture errors that commonly arise in practice but cannot be detected by CFDs and CINDs. We establish two sets of results for central technical problems associated with CFD^{p}s and CIND^{p}s. (a) One concerns the satisfiability and implication problems for CFD^{p}s and CIND^{p}s, taken separately or together. These are important for, *e.g.,* deciding whether data quality rules are dirty themselves, and for removing redundant rules. We show that despite the increased expressive power, the static analyses of CFD^{p}s and CIND^{p}s retain the same complexity as their CFDs and CINDs counterparts. (b) The other concerns validation of CFD^{p}s and CIND^{p}s. We show that given a set Σ of CFD^{p}s and CIND^{p}s on a database D, a set of SQL queries can be automatically generated that, when evaluated against D, return all tuples in D that violate some dependencies in Σ. This provides commercial DBMS with an immediate capability to detect errors based on CFD^{p}s and CIND^{p}s.

Key words: functional dependency, inclusion dependency, data quality

1 Introduction

Extensions of functional dependencies (FDs) and inclusion dependencies (INDs), known as *conditional functional dependencies* (CFDs [14]) and *conditional inclusion dependencies* (CINDs [8]), respectively, have recently been proposed for improving data quality. These extensions enforce patterns of semantically related data values, and detect errors as violations of the dependencies. Conditional dependencies are able to capture more inconsistencies than FDs and INDs [14, 8].

Conditional dependencies specify constant patterns in terms of equality (=). In practice, however, the semantics of data often needs to be specified in terms of other predicates such as $\neq, <, \leq, >$ and $\geq$, as illustrated by the example below.

Example 1. An online store maintains a database of two relations: (a) item for items sold by the store, and (b) tax for the sale tax rates for the items, except artwork, in various states. The relations are specified by the following schemas:

item (id: string, name: string, type: string, price: float, shipping: float, sale: bool, state: string)
tax (state: string, rate: float)

	id	name	type	price	shipping	sale	state
t_1:	b1	Harry Potter	book	25.99	0	T	WA
t_2:	c1	Snow White	CD	9.99	2	F	NY
t_3:	b2	Catch-22	book	34.99	20	F	DL
t_4:	a1	Sunflowers	art	5m	500	F	DL

(a) An item relation

	state	rate
t_5:	PA	6
t_6:	NY	4
t_7:	DL	0
t_8:	NJ	3.5

(b) tax rates

Fig. 1. Example instance D_0 of item and tax

where each item is specified by its id, name, type (*e.g.,* book, CD), price, shipping fee, the state to which it is shipped, and whether it is on sale. A tax tuple specifies the sale tax rate in a state. An instance D_0 of item and tax is shown in Fig. 1.

One wants to specify dependencies on the relations as data quality rules to detect errors in the data, such that inconsistencies emerge as violations of the dependencies. Traditional dependencies (FDs, INDs; see, *e.g.,* [1]) and conditional dependencies (CFDs, CINDs [14, 8]) on the data include the following:

cfd_1: item (id $\rightarrow$ name, type, price, shipping, sale)
cfd_2: tax (state $\rightarrow$ rate)
cfd_3: item (sale = 'T' $\rightarrow$ shipping = 0)

These are CFDs: (a) cfd_1 assures that the id of an item uniquely determines the name, type, price, shipping, sale of the item; (b) cfd_2 states that state is a key for tax, *i.e.,* for each state there is a unique sale tax rate; and (c) cfd_3 is to ensure that for any item tuple t, if t[sale] = 'T' then t[shipping] must be 0; *i.e.,* the store provides free shipping for items on sale. Here cfd_3 is specified in terms of patterns of semantically related data values, namely, sale = 'T' and shipping = 0. It is to hold only on item tuples that match the pattern sale = 'T'. In contrast, cfd_1 and cfd_2 are traditional FDs without constant patterns, a special case of CFDs. One can verify that no sensible INDs or CINDs can be defined across item and tax.

Note that D_0 of Fig. 1 satisfies cfd_1, cfd_2 and cfd_3. That is, when these dependencies are used as data quality rules, no errors are found in D_0.

In practice, the shipment fee of an item is typically determined by the price of the item. Moreover, when an item is on sale, the price of the item is often in a certain range. Furthermore, for any item sold by the store to a customer in a state, if the item is *not* artwork, then one expects to find the sale tax rate in the state from the tax table. These semantic relations cannot be expressed as CFDs of [14] or CINDs of [8], but can be expressed as the following dependencies:

pfd_1: item (sale = 'F' & price $\leq 20 \rightarrow$ shipping = 3)
pfd_2: item (sale = 'F' & price > 20 & price $\leq 40 \rightarrow$ shipping = 6)
pfd_3: item (sale = 'F' & price $> 40 \rightarrow$ shipping = 10)
pfd_4: item (sale = 'T' $\rightarrow$ price ≥ 2.99 & price < 9.99)
pind_1: item (state; type $\neq$ 'art') $\subseteq$ tax (state; nil)

Here pfd_2 states that for any item tuple, if it is not on sale and its price is in the range $(20, 40]$, then its shipment fee must be 6; similarly for pfd_1 and pfd_3. These dependencies extend CFDs [14] by specifying patterns of semantically related data values in terms of predicates $<, \leq, >$, and $\geq$. Similarly, pfd_4 assures that for any item tuple, if it is on sale, then its price must be in the range $[2.99, 9.99)$. Dependency pind_1 extends CINDs [8] by specifying patterns with $\neq$: for any item

tuple t, if $t[\mathsf{type}]$ is *not* artwork, then there must exist a tax tuple t' such that $t[\mathsf{state}] = t'[\mathsf{state}]$, *i.e.,* the sale tax of the item can be found from the tax relation.

Using pfd_1–pfd_4 and pind_1 as data quality rules, we find that D_0 of Fig. 1 is *not* clean. Indeed, (a) t_2 violates pfd_1: its price is less than 20, but its shipping fee is 2 rather than 3; similarly, t_3 violates pfd_2, and t_4 violates pfd_3. (b) Tuple t_1 violates pfd_4: it is on sale but its price is not in the range $[2.99, 9.99)$. (c) The database D_0 also violates pind_1: t_1 is not artwork, but its state cannot find a match in the tax relation, *i.e.,* no tax rate for WA is found in D_0. □

None of pfd_1–pfd_4 and pind_1 can be expressed as FDs or INDs [1], which do not allows constants, or as CFDs [14] or CINDs [8], which specify patterns with equality (=) only. While there have been extensions of CFDs [7, 18], none of these allows dependencies to be specified with patterns on data values in terms of built-in predicates $\neq, <, \leq, >$ or $\geq$. To the best of our knowledge, no previous work has studied extensions of CINDs (see Section 6 for detailed discussions).

These highlight the need for extending CFDs and CINDs to capture errors commonly found in real-life data. While one can consider arbitrary extensions, it is necessary to strike a balance between the expressive power of the extensions and their complexity. In particular, we want to be able to reason about data quality rules expressed as extended CFDs and CINDs. Furthermore, we want to have effective algorithms to detect inconsistencies based on these extensions.

Contributions. This paper proposes a natural extension of CFDs and CINDs, provides complexity bounds for reasoning about the extension, and develops effective SQL-based techniques for detecting errors based on the extension.

(1) We propose two classes of dependencies, denoted by CFD^{p}s and CIND^{p}s, which respectively extend CFDs and CINDs by supporting $\neq, <, \leq, >, \geq$ predicates. For example, all the dependencies we have encountered so far can be expressed as CFD^{p}s or CIND^{p}s. These dependencies are capable of capturing errors in real-world data that cannot be detected by CFDs or CINDs.

(2) We establish complexity bounds for the satisfiability problem and the implication problem for CFD^{p}s and CIND^{p}s, taken separately or together. The satisfiability problem is to determine whether a set Σ of dependencies has a nonempty model, *i.e.,* whether the rules in Σ are consistent themselves. The implication problem is to decide whether a set Σ of dependencies entails another dependency φ, *i.e.,* whether the rule φ is redundant in the presence of the rules in Σ. These are the central technical problems associated with any dependency language.

We show that despite the increased expressive power, CFD^{p}s and CIND^{p}s do not increase the complexity for reasoning about them. In particular, we show that the satisfiability and implication problems remain (a) NP-complete and coNP-complete for CFD^{p}s, respectively, (b) in $O(1)$-time (constant-time) and EXPTIME-complete for CIND^{p}s, respectively, and (c) are undecidable when CFD^{p}s and CIND^{p}s are taken together. These are *the same as* their CFDs and CINDs counterparts. While data with linearly ordered domains often makes our lives harder (see, *e.g.,* [21]), CFD^{p}s and CIND^{p}s do not complicate their static analyses.

(3) We provide SQL-based techniques to detect errors based on CFD^{p}s and CIND^{p}s.

(1) φ_1 = tax (state → rate, T_1)

	state	rate
T_1:	_	_

(2) φ_2 = item (sale → shipping, T_2)

	sale	shipping
T_2:	= T	= 0

(3) φ_3 = item (sale, price →shipping, T_3)

	sale	price	shipping
	= F	> 20	= 6
T_3:	= F	≤ 40	= 6

(4) CFD^p φ_4 = item (sale → price, T_4)

	sale	price
	= T	≥ 2.99
T_4:	= T	< 9.99

Fig. 2. Example CFD^{p}s

Given a set Σ of CFD^{p}s and CIND^{p}s on a database D, we automatically generate a set of SQL queries that, when evaluated on D, find all tuples in D that violate some dependencies in Σ. Further, the SQL queries are independent of the size and cardinality of Σ. No previous work has been studied error detection based on CINDs, not to mention CFD^{p}s and CIND^{p}s taken together. These provide the capability of detecting errors in a single relation (CFD^{p}s) and across different relations (CIND^{p}s) within the immediate reach of commercial DBMS.

Organizations. Sections 2 and 3 introduce CFD^{p}s and CIND^{p}s, respectively. Section 4 establishes complexity bounds for reasoning about CFD^{p}s and CIND^{p}s. Section 5 provides SQL techniques for error detection. Related work is discussed in Section 6, followed by topics for future work in Section 7.

2 Incorporating Built-in Predicates into CFDs

We now define CFD^{p}s, also referred to as *conditional functional dependencies*, by extending CFDs with predicates ($\neq, <, \leq, >, \geq$) in addition to equality ($=$).

Consider a relation schema R defined over a finite set of attributes, denoted by $\mathsf{attr}(R)$. For each attribute $A \in \mathsf{attr}(R)$, its domain is specified in R, denoted as $\mathsf{dom}(A)$, which is either finite (*e.g.,* bool) or infinite (*e.g.,* string). We assume *w.l.o.g.* that a domain is totally ordered if $<, \leq, >$ or $\geq$ is defined on it.

Syntax. A CFD^p φ on R is a pair $R(X \rightarrow Y,\ T_p)$, where (1) X, Y are sets of attributes in $\mathsf{attr}(R)$; (2) $X \rightarrow Y$ is a standard FD, referred to as the FD *embedded in* φ; and (3) T_p is a tableau with attributes in X and Y, referred to as the *pattern tableau* of φ, where for each A in $X \cup Y$ and each tuple $t_p \in T_p$, $t_p[A]$ is either an unnamed variable '_' that draws values from $\mathsf{dom}(A)$, or '$\mathsf{op}\ a$', where op is one of $=, \neq, <, \leq, >, \geq$, and '$a$' is a constant in $\mathsf{dom}(A)$.

If attribute A occurs in both X and Y, we use A_L and A_R to indicate the occurrence of A in X and Y, respectively, and separate the X and Y attributes in a pattern tuple with '$\|$'. We write φ as $(X \rightarrow Y,\ T_p)$ when R is clear from the context, and denote X as $\mathsf{LHS}(\varphi)$ and Y as $\mathsf{RHS}(\varphi)$.

Example 2. The dependencies cfd1–cfd3 and pfd1–pfd4 that we have seen in Example 1 can all be expressed as CFD^{p}s. Figure 2 shows some of these CFD^{p}s: φ_1 (for FD cfd_2), φ_2 (for CFD cfd_3), φ_3 (for pfd_2), and φ_4 (for pfd_4). □

Semantics. Consider CFD^p $\varphi = (R : X \rightarrow Y,\ T_p)$, where $T_p = \{t_{p1}, \ldots, t_{pk}\}$.

A data tuple t of R is said to *match* $\mathsf{LHS}(\varphi)$, denoted by $t[X] \asymp T_p[X]$, if *for each tuple* t_{pi} *in* T_p and *each attribute* A in X, either (a) $t_{pi}[A]$ is the wildcard '_'

(which matches any value in $\mathsf{dom}(A)$), or (b) $t[A]$ op a if $t_{pi}[A]$ is 'op a', where the operator op $(=, \neq, <, \leq, >$ or $\geq)$ is interpreted by its standard semantics. Similarly, the notion that t *matches* $\mathsf{RHS}(\varphi)$ is defined, denoted by $t[Y] \asymp T_p[Y]$.

Intuitively, each pattern tuple t_{pi} specifies a condition via $t_{pi}[X]$, and $t[X] \asymp T_p[X]$ if $t[X]$ satisfies the *conjunction* of all these conditions. Similarly, $t[Y] \asymp T_p[Y]$ if $t[Y]$ matches all the patterns specified by $t_{pi}[Y]$ for all t_{pi} in T_p.

An instance I of R *satisfies* the CFD^p φ, denoted by $I \models \varphi$, if for *each pair* of tuples t_1, t_2 in the instance I, if $t_1[X] = t_2[X] \asymp T_p[X]$, then $t_1[Y] = t_2[Y] \asymp T_p[Y]$. That is, if $t_1[X]$ and $t_2[X]$ are equal and in addition, they both match the pattern tableau $T_p[X]$, then $t_1[Y]$ and $t_2[Y]$ must also be equal to each other and they both match the pattern tableau $T_p[Y]$.

Observe that φ is imposed only on the subset of tuples in I that match $\mathsf{LHS}(\varphi)$, rather than on the entire I. For all tuples t_1, t_2 in this subset, if $t_1[X] = t_2[X]$, then (a) $t_1[Y] = t_2[Y]$, *i.e.,* the semantics of the embedded FDs is enforced; and (b) $t_1[Y] \asymp T_p[Y]$, which assures that the *constants* in $t_1[Y]$ match the *constants* in $t_{pi}[Y]$ for all t_{pi} in T_p. Note that here tuples t_1 and t_2 can be the same.

An instance I of R satisfies a set Σ of CFD^{p}s, denoted by $I \models \Sigma$, if $I \models \varphi$ for *each* CFD^p φ in Σ.

Example 3. The instance D_0 of Fig. 1 satisfies φ_1 and φ_2 of Fig. 2, but neither φ_3 nor φ_4. Indeed, tuple t_3 violates (*i.e.,* does not satisfy) φ_3, since $t_3[\mathsf{sale}] =$ 'F' *and* $20 < t_3[\mathsf{price}] \leq 40$, but $t_3[\mathsf{shipping}]$ is 20 instead of 6. Note that t_3 matches $\mathsf{LHS}(\varphi_3)$ since it satisfies the condition specified by the *conjunction* of the pattern tuples in T_3. Similarly, t_1 violates φ_4, since $t_1[\mathsf{sale}] =$ 'T' but $t_1[\mathsf{price}] > 9.99$. Observe that while it takes two tuples to violate a standard FD, a single tuple may violate a CFD^p. □

Special cases. (1) A standard FD $X \to Y$ [1] can be expressed as a CFD $(X \to Y,\ T_p)$ in which T_p contains a single tuple consisting of '_' only, without constants. (2) A CFD $(X \to Y,\ T_p)$ [14] with $T_p = \{t_{p1}, \ldots, t_{pk}\}$ can be expressed as a set $\{\varphi_1, \ldots, \varphi_k\}$ of CFD^{p}s such that for $i \in [1, k]$, $\varphi_i = (X \to Y,\ T_{pi})$, where T_{pi} contains a single pattern tuple t_{pi} of T_p, with equality $(=)$ only. For example, φ_1 and φ_2 in Fig. 2 are CFD^{p}s representing FD cfd2 and CFD cfd3 in Example 1, respectively. Note that all data quality rules in [10, 18] can be expressed as CFD^{p}s.

3 Incorporating Built-in Predicates into CINDs

Along the same lines as CFD^{p}s, we next define CIND^{p}s, also referred to as *conditional inclusion dependencies*. Consider two relation schemas R_1 and R_2.

Syntax. A CIND^p ψ is a pair $(R_1[X;\ X_p] \subseteq R_2[Y;\ Y_p],\ T_p)$, where (1) X, X_p and Y, Y_p are lists of attributes in $\mathsf{attr}(R_1)$ and $\mathsf{attr}(R_2)$, respectively; (2) $R_1[X] \subseteq R_2[Y]$ is a standard IND, referred to as the IND *embedded* in ψ; and (3) T_p is a tableau, called the *pattern tableau* of ψ defined over attributes $X_p \cup Y_p$, and for each A in X_p or Y_p and each pattern tuple $t_p \in T_p$, $t_p[A]$ is either an

(1) $\psi_1 = (\mathsf{item}\ [\mathsf{state};\ \mathsf{type}] \subseteq \mathsf{tax}\ [\mathsf{state};\ \mathsf{nil}],\ T_1)$,

(2) $\psi_2 = (\mathsf{item}\ [\mathsf{state},\ \mathsf{type},\ \mathsf{state}] \subseteq \mathsf{tax}\ [\mathsf{state};\ \mathsf{rate}],\ T_2)$

T_1:

type	nil
$\neq$ art	

T_2:

type	state	rate
$\neq$ art	$=$ DL	$= 0$

Fig. 3. Example CIND^{p}s

unnamed variable '_' that draws values from $\mathsf{dom}(A)$, or 'op a', where op is one of $=, \neq, <, \leq, >, \geq$ and 'a' is a constant in $\mathsf{dom}(A)$.

We denote $X \cup X_p$ as $\mathsf{LHS}(\psi)$ and $Y \cup Y_p$ as $\mathsf{RHS}(\psi)$, and separate the X_p and Y_p attributes in a pattern tuple with '$\|$'. We use nil to denote an *empty* list.

Example 4. Figure 3 shows two example CIND^{p}s: ψ_1 expresses pind_1 of Example 1, and ψ_2 refines ψ_1 by stating that for any item tuple t_1, if its type is not art and its state is DL, then there must be a tax tuple t_2 such that its state is DL and rate is 0, *i.e.,* ψ_2 assures that the sale tax rate in Delaware is 0. □

Semantics. Consider CIND^p $\psi = (R_1[X;\ X_p] \subseteq R_2[Y;\ Y_p],\ T_p)$. An instance (I_1, I_2) of (R_1, R_2) *satisfies* the CIND^p ψ, denoted by $(I_1, I_2) \models \psi$, iff for *each* tuple $t_1 \in I_1$, if $t_1[X_p] \asymp T_p[X_p]$, then there *exists* a tuple $t_2 \in I_2$ such that $t_1[X] = t_2[Y]$ and moreover, $t_2[Y_p] \asymp T_p[Y_p]$.

That is, if $t_1[X_p]$ matches the pattern tableau $T_p[X_p]$, then ψ requires the existence of t_2 such that (1) $t_1[X] = t_2[Y]$ as required by the standard IND embedded in ψ; and (2) $t_2[Y_p]$ must match the pattern tableau $T_p[Y_p]$. In other words, ψ is "conditional" since its embedded IND is applied only to the subset of tuples in I_1 that match $T_p[X_p]$, and moreover, the pattern $T_p[Y_p]$ is enforced on the tuples in I_2 that match those tuples in I_1. As remarked in Section 2, the pattern tableau T_p specifies the *conjunction* of patterns of all tuples in T_p.

Example 5. The instance D_0 of item and tax in Fig. 1 violates CIND^p ψ_1. Indeed, tuple t_1 in item *matches* $\mathsf{LHS}(\psi_1)$ since $t_1[\mathsf{type}] \neq$ 'art', but there is no tuple t in tax such that $t[\mathsf{state}] = t_1[\mathsf{state}] =$ 'WA'. In contrast, D_0 satisfies ψ_2. □

We say that a database D satisfies a set Σ of CINDs, denoted by $D \models \Sigma$, if $D \models \varphi$ for each $\varphi \in \Sigma$.

Safe CIND^{p}s. We say a CIND^p $(R_1[X;\ X_p] \subseteq R_2[Y;\ Y_p],\ T_p)$ is *unsafe* if there exist pattern tuples t_p, t'_p in T_p such that either (a) there exists $B \in Y_p$, such that $t_p[B]$ and $t'_p[B]$ are not satisfiable when taken together, or (b) there exist $C \in Y, A \in X$ such that A corresponds to B in the IND and $t_p[C]$ and $t'_p[A]$ are not satisfiable when taken together; *e.g.,* $t_p[\mathsf{price}] = 9.99$ and $t'_p[\mathsf{price}] \geq 19.99$.

Obviously unsafe CIND^{p}s do not make sense: there exist no nonempty database that satisfies unsafe CIND^{p}s. It takes $O(|T_p|^2)$-time in the size $|T_p|$ of T_p to decide whether a CIND^p is unsafe. Thus in the sequel we consider safe CIND^p only.

Special cases. Observe that (1) a standard CIND $(R_1[X] \subseteq R_2[Y])$ can be expressed as a CIND^p $(R_1[X;\ \mathsf{nil}] \subseteq R_2[Y;\ \mathsf{nil}],\ T_p)$ such that T_p is simply a *empty* set; and (2) a CIND $(R_1[X;\ X_p] \subseteq R_2[Y;\ Y_p],\ T_p)$ with $T_p = \{t_{p1}, \ldots, t_{pk}\}$ can be expressed as a set $\{\psi_1, \ldots, \psi_k\}$ of CIND^{p}s, where for $i \in [1, k]$, $\psi_i = (R_1[X;\ X_p] \subseteq R_2[Y;\ Y_p],\ T_{pi})$ such that T_{pi} consists of a single pattern tuple t_{pi} of T_p defined in terms of equality $(=)$ only.

4 Reasoning about CFD^{p}s and CIND^{p}s

The satisfiability problem and the implication problem are the two central technical questions associated with any dependency languages. In this section we investigate these problems for CFD^{p}s and CIND^{p}s, separately and taken together.

4.1 The Satisfiability Analysis

The satisfiability problem is to determine, given a set Σ of constraints, whether there exists a *nonempty* database that satisfies Σ.

The satisfiability analysis of conditional dependencies is not only of theoretical interest, but is also important in practice. Indeed, when CFD^{p}s and CIND^{p}s are used as data quality rules, this analysis helps one check whether the rules make sense themselves. The need for this is particularly evident when the rules are manually designed or discovered from various datasets [10, 18, 15].

The satisfiability analysis of CFD^{p}s. Given any FDs, one does not need to worry about their satisfiability since any set of FDs is always satisfiable. However, as observed in [14], for a set Σ of CFDs on a relational schema R, there may not exist a *nonempty* instance I of R such that $I \models \Sigma$. As CFDs are a special case of CFD^{p}s, the same problem exists when it comes to CFD^{p}s.

Example 6. Consider CFD^p $\varphi = (R : A \rightarrow B,\ T_p)$ such that $T_p = \{(\_ \parallel = a), (\_ \parallel \neq a)\}$. Then there exists no *nonempty* instance I of R that satisfies φ. Indeed, for any tuple t of R, φ requires that both $t[B] = a$ and $t[B] \neq a$. □

This problem is already NP-complete for CFDs [14]. Below we show that it has the same complexity for CFD^{p}s despite their increased expressive power.

Proposition 1. *The satisfiability problem for* CFD^p*s is* NP*-complete.* □

Proof sketch: The lower bound follows from the NP-hardness of their CFDs counterparts [14], since CFDs are a special case of CFD^{p}s. The upper bound is verified by presenting an NP algorithm that, given a set Σ of CFD^{p}s defined on a relation schema R, determines whether Σ is satisfiable. □

It is known [14] that the satisfiability problem for CFDs is in PTIME when the CFDs considered are defined over attributes that have an infinite domain, *i.e.,* in the absence of finite domain attributes. However, this is no longer the case for CFD^{p}s. This tells us that the increased expressive power of CFD^{p}s does take a toll in this special case. It should be remarked that while the proof of Proposition 1 is an extension of its counterpart in [14], the result below is new.

Theorem 2. *In the absence of finite domain attributes, the satisfiability problem for* CFD^p*s remains* NP*-complete.* □

Proof sketch: The problem is in NP by Proposition 1. Its NP-hardness is shown by reduction from the 3SAT problem, which is NP-complete (cf. [17]). □

The satisfiability analysis of CIND^{p}s. Like FDs, one can specify arbitrary INDs or CINDs without worrying about their satisfiability. Below we show that CIND^{p}s also have this property, by extending the proof of its counterpart in [8].

Proposition 3. *Any set Σ of* CIND^p*s is always satisfiable.* □

Proof sketch: Given a set Σ of CIND^{p}s over a database schema $\mathcal{R}$, one can always construct a *nonempty* instance D of $\mathcal{R}$ such that $D \models \Sigma$. □

The satisfiability analysis of CFD^{p}s and CIND^{p}s. The satisfiability problem for CFDs and CINDs taken together is undecidable [8]. Since CFD^{p}s and CIND^{p}s subsume CFDs and CINDs, respectively, from these we immediately have:

Corollary 4. *The satisfiability problem for* CFD^p*s and* CIND^p*s is undecidable.*□

4.2 The Implication Analysis

The implication problem is to determine, given a set Σ of dependencies and another dependency ϕ, whether or not Σ entails ϕ, denoted by $\Sigma \models \phi$. That is, whether or not for all databases D, if $D \models \Sigma$ then $D \models \phi$.

The implication analysis helps us remove redundant data quality rules, and thus improve the performance of error detection and repairing based on the rules.

Example 7. The CFD^{p}s of Fig. 2 imply CFD^{p}s $\varphi = \mathsf{item}\ (\mathsf{sale}, \mathsf{price} \rightarrow \mathsf{shipping}, T)$, where T consists of a single pattern tuple (sale ='F', $\mathsf{price} = 30 \parallel \mathsf{shipping} = 6$). Thus in the presence of the CFD^{p}s of Fig. 2, φ is redundant. □

The implication analysis of CFD^{p}s. We first show that the implication problem for CFD^{p}s retains the same complexity as their CFDs counterpart. The result below is verified by extending the proof of its counterpart in [14].

Proposition 5. *The implication problem for* CFD^p*s is co*NP*-complete.* □

Proof sketch: The lower bound follows from the coNP-hardness of their CFDs counterpart [14], since CFDs are a special case of CFD^{p}s. The coNP upper bound is verified by presenting an NP algorithm for its complement problem, *i.e.,* the problem for determining whether $\Sigma \not\models \varphi$. □

Similar to the satisfiability analysis, it is known [14] that the implication analysis of CFDs is in PTIME when the CFDs are defined only with attributes that have an infinite domain. Analogous to Theorem 2, the result below shows that this is no longer the case for CFD^{p}s, which does not find a counterpart in [14].

Theorem 6. *In the absence of finite domain attributes, the implication problem for* CFD^p*s remains co*NP*-complete.* □

Proof sketch: It is in coNP by Proposition 5. The coNP-hardness is shown by reduction from the 3SAT problem to its complement problem, *i.e.,* the problem for determining whether $\Sigma \not\models \varphi$. □

The implication analysis of CIND^{p}s. We next show that CIND^{p}s do not make their implication analysis harder. This is verified by extending the proof of their CINDs counterpart given in [8].

Proposition 7. *The implication problem for* CIND^p*s is* EXPTIME*-complete.* □

Σ	General setting		Infinite domain only	
	Satisfiability	Implication	Satisfiability	Implication
CFDs [14]	NP-complete	coNP-complete	PTIME	PTIME
CFD^{p}s	NP-complete	coNP-complete	NP-complete	coNP-complete
CINDs [8]	$O(1)$	EXPTIME-complete	$O(1)$	PSPACE-complete
CIND^{p}s	$O(1)$	EXPTIME-complete	$O(1)$	EXPTIME-complete
CFDs + CINDs [8]	undecidable	undecidable	undecidable	undecidable
CFD^{p}s + CIND^{p}s	undecidable	undecidable	undecidable	undecidable

Table 1. Summary of complexity results

Proof sketch: The implication problem for CINDs is EXPTIME-hard [8]. The lower bound carries over to CIND^{p}s since CIND^{p}s subsume CINDs. The EXPTIME upper bound is shown by presenting an EXPTIME algorithm that, given a set $\Sigma \cup \{\psi\}$ of CIND^{p}s over a database schema $\mathcal{R}$, determines whether $\Sigma \models \psi$. □

It is known [8] that the implication problem is PSPACE-complete for CINDs defined with infinite-domain attributes. Similar to Theorem 6, below we present a new result showing that this no longer holds for CIND^{p}s.

Theorem 8. *In the absence of finite domain attributes, the implication problem for* CIND^p*s remains* EXPTIME*-complete.* □

Proof sketch: The EXPTIME upper bound follows from Proposition 7. The EXPTIME-hardness is shown by reduction from the implication problem for CINDs in the general setting, in which finite-domain attributes may be present; the latter is known to be EXPTIME-complete [8]. □

The implication analysis of CFD^{p}s and CIND^{p}s. When CFD^{p}s and CIND^{p}s are taken together, their implication analysis is beyond reach in practice. This is not surprising since the implication problem for FDs and INDs is already undecidable [1]. Since CFD^{p}s and CIND^{p}s subsume FDs and INDs, respectively, from the undecidability result for FDs and INDs, the corollary below follows immediately.

Corollary 9. *The implication problem for* CFD^p*s and* CIND^p*s is undecidable.* □

Summary. The complexity bounds for reasoning about CFD^{p}s and CIND^{p}s are summarized in Table 1. To give a complete picture we also include in Table 1 the complexity bounds for the static analyses of CFDs and CINDs, taken from [14, 8]. The results shown in Table 1 tell us the following.

(a) Despite the increased expressive power, CFD^{p}s and CIND^{p}s do not complicate the static analyses: the satisfiability and implication problems for CFD^{p}s and CIND^{p}s have the same complexity bounds as their counterparts for CFDs and CINDs, taken separately or together.

(b) In the special case when CFD^{p}s and CIND^{p}s are defined with infinite-domain attributes only, however, the static analyses of CFD^{p}s and CIND^{p}s do not get simpler, as opposed to their counterparts for CFDs and CINDs. That is, in this special case the increased expressive power of CFD^{p}s and CIND^{p}s comes at a price.

5 Validation of CFD^{p}s and CIND^{p}s

If CFD^{p}s and CIND^{p}s are to be used as data quality rules, the first question we have to settle is how to effectively detect errors and inconsistencies as violations of these dependencies, by leveraging functionality supported by commercial DBMS. More specifically, consider a database schema $\mathcal{R} = (R_1, \ldots, R_n)$, where R_i is a relation schema for $i \in [1, n]$. The error detection problem is stated as follows.

The *error detection problem* is to find, given a set Σ of CFD^{p}s and CIND^{p}s defined on $\mathcal{R}$, and a database instance $D = (I_1, \ldots, I_n)$ of $\mathcal{R}$ as input, the subset $(I'_1, \ldots, I'_n)$ of D such that for each $i \in [1, n]$, $I'_i \subseteq I_i$ and each tuple in I'_i violates at least one CFD^p or CIND^p in Σ. We denote the set as $\mathsf{vio}(D, \Sigma)$, referred to it as *the violation set* of D *w.r.t.* Σ.

In this section we develop SQL-based techniques for error detection based on CFD^{p}s and CIND^{p}s. The main result of the section is as follows.

Theorem 10. *Given a set Σ of* CFD^p*s and* CIND^p*s defined on $\mathcal{R}$ and a database instance D of $\mathcal{R}$, where $\mathcal{R} = (R_1, \ldots, R_n)$, a set of* SQL *queries can be automatically generated such that (a) the collection of the answers to the* SQL *queries in D is $\mathsf{vio}(D, \Sigma)$, (b) the number and size of the set of* SQL *queries depend only on the number n of relations and their arities in $\mathcal{R}$, regardless of Σ.* □

We next present the main techniques for the query generation method. Let $\Sigma^i_{\mathsf{cfd^p}}$ be the set of all CFD^{p}s in Σ defined on the same relation schema R_i, and $\Sigma^{(i,j)}_{\mathsf{cind^p}}$ the set of all CIND^{p}s in Σ from R_i to R_j, for $i, j \in [1, n]$. We show the following. (a) The violation set $\mathsf{vio}(D, \Sigma^i_{\mathsf{cfd^p}})$ can be computed by *two* SQL queries. (b) Similarly, $\mathsf{vio}(D, \Sigma^{(i,j)}_{\mathsf{cind^p}})$ can be computed by a *single* SQL query. (c) These SQL queries encode pattern tableaux of CFD^{p}s (CIND^{p}s) with data tables, and hence their sizes are independent of Σ. From these Theorem 10 follows immediately.

5.1 Encoding CFD^{p}s and CIND^{p}s with Data Tables

We first show the following, by extending the encoding of [14, 7]. (a) The pattern tableaux of all CFD^{p}s in $\Sigma^i_{\mathsf{cfd^p}}$ can be encoded with *three data tables*, and (b) the pattern tableaux of all CIND^{p}s in $\Sigma^{(i,j)}_{\mathsf{cind^p}}$ can be represented as *four data tables*, no matter how many dependencies are in the sets and how large they are.

Encoding CFD^{p}s. We encode all pattern tableaux in $\Sigma^i_{\mathsf{cfd^p}}$ with three tables enc_L, enc_R and $\mathsf{enc}_{\neq}$, where enc_L (resp. enc_R) encodes the non-negation ($=, <, \leq, >, \geq$) patterns in LHS (resp. RHS), and $\mathsf{enc}_{\neq}$ encodes those negation ($\neq$) patterns. More specifically, we associate a unique id cid with each CFD^{p}s in $\Sigma^i_{\mathsf{cfd^p}}$, and let enc_L consist of the following attributes: (a) cid, (b) each attribute A appearing in the LHS of some CFD^{p}s in $\Sigma^i_{\mathsf{cfd^p}}$, and (b) its four companion attributes $A_>$, $A_\geq$, $A_<$, and $A_\leq$. That is, for each attribute, there are five columns in enc_L, one for each non-negation operator. Similarly, enc_R is defined. We use an $\mathsf{enc}_{\neq}$ tuple to encode a pattern $A \neq c$ in a CFD^p, consisting of cid, att, pos, and val, encoding the CFD^p id, the attribute A, the position ('LHS' or 'RHS'),

(1) enc_L

cid	sale	price	$\text{price}_>$	$\text{price}_\leq$
2	T	null	null	null
3	F	_	20	40
4	T	null	null	null

(2) enc_R

cid	shipping	price	$\text{price}_\geq$	$\text{price}_<$
2	0	null	null	null
3	6	null	null	null
4	null	_	2.99	9.99

(3) $\mathsf{enc}_{\neq}$

cid	pos	att	val

Fig. 4. Encoding example of CFD^{p}s

and the constant c, respectively. Note that the arity of enc_L (enc_R) is bounded by $5 * |R_i| + 1$, where $|R_i|$ is the arity of R_i, and the arity of $\mathsf{enc}_{\neq}$ is 4.

Before we populate these tables, let us first describe a preferred form of CFD^{p}s that would simplify the analysis to be given. Consider a CFD^p $\varphi = R(X \rightarrow Y,\ T_p)$. If φ is not satisfiable we can simply drop it from Σ. Otherwise it is equivalent to a CFD^p $\varphi' = R(X \rightarrow Y,\ T'_p)$ such that for any pattern tuples t_p, t'_p in T'_p and for any attribute A in $X \cup Y$, (a) if $t_p[A]$ is $\mathsf{op}\ a$ and $t'_p[A]$ is $\mathsf{op}\ b$, where op is not $\neq$, then $a = b$, (b) if $t_p[A]$ is '_' then so is $t'_p[A]$. That is, for each non-negation op (resp. _), there is a *unique* constant a such that $t_p[A] =$ '$\mathsf{op}\ a$' (resp. $t_p[A] =$ _) is the only op (resp. _) pattern appearing in the A column of T'_p. We refer to $t_p[A]$ as $T'_p(\mathsf{op}, A)$ (resp. $T'_p(\_, A)$), and consider *w.l.o.g.* CFD^{p}s of this form only. Note that there are possibly multiple $t_p[A] \neq c$ patterns in T'_p,

We populate enc_L, enc_R and $\mathsf{enc}_{\neq}$ as follows. For each CFD^p $\varphi = R(X \rightarrow Y,\ T_p)$ in $\Sigma^i_{\mathsf{cfd^p}}$, we generate a distinct cid id_φ for it, and do the following.

- Add a tuple t_1 to enc_L such that (a) $t[\mathsf{cid}] = \mathsf{id}_\varphi$; (b) for each $A \in X$, $t[A] =$ _ if $T'_p(\_, A)$ is '_', and for each non-negation predicate op, $t[A_{\mathsf{op}}] =$ 'a' if $T'_p(\mathsf{op}, A)$ is '$\mathsf{op}\ a$'; (c) we let $t[B] =$ 'null' for all other attributes B in enc_L.
- Similarly add a tuple t_2 to enc_R for attributes in Y.
- For each attribute $A \in X \cup Y$ and each $\neq a$ pattern in $T_p[A]$, add a tuple t to $\mathsf{enc}_{\neq}$ such that $t[\mathsf{cid}] = \mathsf{id}_\varphi$, $t[\mathsf{att}] =$ 'A', $t[\mathsf{val}] =$ 'a', and $t[\mathsf{pos}] =$ 'LHS' (resp. $t[\mathsf{pos}] =$ 'RHS') if attribute A appears in X (resp. Y).

Example 8. Recall from Fig. 2 CFD^{p}s φ_2, φ_3 and φ_4 defined on relation item. The three CFD^{p}s are encoded with tables shown in Fig. 4: (a) enc_L consists of attributes: cid, sale, price, $\text{price}_>$ and $\text{price}_\leq$; (b) enc_R consists of cid, $\mathsf{shipping}$, price, $\text{price}_\geq$ and $\text{price}_<$; those attributes in a table with only 'null' pattern values do not contribute to error detection, and are thus omitted; (c) $\mathsf{enc}_{\neq}$ is empty since all these CFD^{p}s have no negation patterns. One can easily reconstruct these CFD^{p}s from tables enc_L, enc_R and $\mathsf{enc}_{\neq}$ by collating tuples based on cid. □

Encoding CIND^{p}s. All CIND^{p}s in $\Sigma^{(i,j)}_{\mathsf{cind^p}}$ can be encoded with four tables enc, enc_L, enc_R and $\mathsf{enc}_{\neq}$. Here enc_L (resp. enc_R) and $\mathsf{enc}_{\neq}$ encode non-negation patterns on relation R_i (resp. R_j) and negation patterns on relations R_i or R_j, respectively, along the same lines as their counterparts for CFD^{p}s. We use enc to encode the INDs *embedded* in CIND^{p}s, which consists of the following attributes: (1) cid representing the id of a CIND^p, and (2) those X attributes of R_i and Y attributes of R_j appearing in some CIND^{p}s in $\Sigma^{(i,j)}_{\mathsf{cind^p}}$. Note that the number of attributes in enc is bounded by $|R_i| + |R_j| + 1$, where $|R_i|$ is the arity of R_i.

(1) enc

cid	$\mathsf{state_L}$	$\mathsf{state_R}$
1	1	1
2	1	1

(2) enc_L

cid	type	state
1	_	null
2	_	DL

(3) enc_R

cid	rate
1	null
2	0

(4) $\mathsf{enc}_{\neq}$

cid	pos	att	val
1	LHS	type	art
2	LHS	type	art

Fig. 5. Encoding example of CIND^{p}s

For each CIND^p $\psi = (R_i[A_1 \dots A_m;\ X_p] \subseteq R_j[B_1 \dots B_m;\ Y_p],\ T_p)$ in $\Sigma^{(i,j)}_{\mathsf{cind^p}}$, we generate a distinct cid id_ψ for it, and do the following.

- Add tuples t_1 and t_2 to enc_L and enc_R based on attributes X_p and Y_p, respectively, along the same lines as their CFD^p counterpart.
- Add tuples to $\mathsf{enc}_{\neq}$ in the same way as their CFD^p counterparts.
- Add tuple t to enc such that $t[\mathsf{cid}] = \mathsf{id}_\psi$. For each $k \in [1, m]$, let $t[A_k] = t[B_k] = k$, and $t[A]$ = 'null' for the rest attributes A of enc.

Example 9. Figure 4 shows the coding of CIND^{p}s ψ_1 and ψ_2 given in Fig. 3. We use $\mathsf{state_L}$ and $\mathsf{state_R}$ in enc to denote the occurrences of attribute state in item and tax, respectively. In tables enc_L and enc_R, attributes with only 'null' patterns are omitted, for the same reason as for CFD^{p}s mentioned above. □

Putting these together, it is easy to verify that at most $O(n^2)$ data tables are needed to encode dependencies in Σ, regardless of the size of Σ. Recall that n is the number of relations in database $\mathcal{R}$.

5.2 SQL-based Detection Methods

We next show how to generate SQL queries based on the encoding above. For each $i \in [1, n]$, we generate *two* SQL queries that, when evaluated on the I_i table of D, find $\mathsf{vio}(D, \Sigma^i_{\mathsf{cfd^p}})$. Similarly, for each $i, j \in [1, n]$, we generate a *single* SQL query $Q_{(i,j)}$ that, when evaluated on (I_i, I_j) of D, returns $\mathsf{vio}(D, \Sigma^{(i,j)}_{\mathsf{cind^p}})$. Putting these query answers together, we get $\mathsf{vio}(D, \Sigma)$, the violation set of D *w.r.t.* Σ.

Below we show how the SQL query $Q_{(i,j)}$ is generated for validating CIND^{p}s in $\Sigma^{(i,j)}_{\mathsf{cind^p}}$), which has not been studied by previous work. For the lack of space we omit the generation of detection queries for CFD^{p}s, which is an extension of the SQL techniques for CFDs discussed in [14, 7].

The query $Q_{(i,j)}$ for the validation of $\Sigma^{(i,j)}_{\mathsf{cind^p}}$ is given as follows, which capitalizes on the data tables enc, enc_L, enc_R and $\mathsf{enc}_{\neq}$ that encode CIND^{p}s in $\Sigma^{(i,j)}_{\mathsf{cind^p}}$.

```
select R_i.*
from R_i, enc_L L, enc_≠ N
where R_i.X ≍ L and R_i.X ≍ N and not exists (
        select R_j.*
        from R_j, enc H, enc_R R, enc_≠ N
        where R_i.X = R_j.Y and L.cid = R.cid and L.cid = H.cid and
              R_j.Y ≍ R and R_j.Y ≍ N)
```

Here (1) $X = \{\mathsf{A_1}, \dots, \mathsf{A_{m1}}\}$ and $Y = \{\mathsf{B_1}, \dots, \mathsf{B_{m2}}\}$ are the sets of attributes of R_i and R_j appearing in $\Sigma^{(i,j)}_{\mathsf{cind^p}}$, respectively; (2) $R_i.X \asymp L$ is the conjunction of

$L.A_k$ **is null or** $R_i.A_k = L.A_k$ **or** ($L.A_k$ = '_'
and ($L.A_{i_>}$ **is null or** $R_i.A_k > L.A_{i_>}$) **and** ($L.A_{i_\geq}$ **is null or** $R_i.A_k \geq L.A_{k_\geq}$)
and ($L.A_{k_<}$ **is null or** $R_i.A_k < L.A_{k_<}$) **and** ($L.A_{i_\leq}$ **is null or** $R_i.A_k \leq L.A_{i_\leq}$))

for $k \in [1, m_1]$; (3) $R_j.Y \asymp R$ is defined similarly for attributes in Y; (4) $R_i.X \asymp N$ is a shorthand for the conjunction below, for $k \in [1, m_1]$:

not exists (**select** $*$ **from** N **where** $L.\mathsf{cid} = N.\mathsf{cid}$ **and** $N.\mathsf{pos}$ = 'LHS' **and**
$N.\mathsf{att}$ = 'A_k' **and** $R_i.A_k = N.\mathsf{val}$);

(5) $R_j.Y \asymp N$ is defined similarly, but with $N.\mathsf{pos}$ = 'RHS' ; (6) $R_i.X = R_j.Y$ represents the following: for each A_k ($k \in [1, m_1]$) and each B_l ($l \in [1, m_2]$), ($H.A_k$ **is null or** $H.B_l$ **is null or** $H.B_l \neq H.A_k$ **or** $R_i.A_k = R_j.B_l$).

Intuitively, (1) $R_i.X \asymp L$ **and** $R_i.X \asymp N$ ensure that the R_i tuples selected match the LHS patterns of some CIND^{p}s in $\Sigma^{(i,j)}_{\mathsf{cind^p}}$; (2) $R_j.Y \asymp R$ **and** $R_j.Y \asymp N$ check the corresponding RHS patterns of these CIND^{p}s on R_j tuples; (3) $R_i.X = R_j.Y$ enforces the *embedded* INDs; (4) $L.\mathsf{cid} = R.\mathsf{cid}$ **and** $L.\mathsf{cid} = H.\mathsf{cid}$ assure that the LHS and RHS patterns in the same CIND^p are correctly collated; and (5) **not exists** in Q ensures that the R_i tuples selected violate CIND^{p}s in $\Sigma^{(i,j)}_{\mathsf{cind^p}}$.

Example 10. Using the coding of Fig. 5, an SQL query Q for checking CIND^{p}s ψ_1 and ψ_2 of Fig. 3 is given as follows:

select $R_1.*$ **from** item R_1, enc_L L, $\mathsf{enc}_{\neq}$ N
where ($L.\mathsf{type}$ **is null or** $R_1.\mathsf{type} = L.\mathsf{type}$ **or** $L.\mathsf{type}$ = '_') **and not exist** (
select * from N
where $N.\mathsf{cid} = L.\mathsf{cid}$ **and** $N.\mathsf{pos}$ = 'LHS' **and** $N.\mathsf{att}$ = 'type')
and ($L.\mathsf{state}$ **is null or** $R_1.\mathsf{state} = L.\mathsf{state}$ **or** $L.\mathsf{state}$ = '_') **and not exist** (
select * from N
where $N.\mathsf{cid} = L.\mathsf{cid}$ **and** $N.\mathsf{pos}$ = 'LHS' **and** $N.\mathsf{att}$ = 'state' **and** $R_1.\mathsf{state}$ =N.val)

and not exists (
select $R_2.*$ **from** tax R_2, enc H, enc_R R
where ($H.\mathsf{state_L}$ **is null or** $H.\mathsf{state_R}$ **is null or** $H.\mathsf{state_L}! = H.\mathsf{state_R}$ **or**
$R_2.\mathsf{state} = R_1.\mathsf{state}$) **and** $L.\mathsf{cid} = H.\mathsf{cid}$ **and** $L.\mathsf{cid} = R.\mathsf{cid}$ **and**
($R.\mathsf{rate}$ **is null or** $R_2.\mathsf{rate} = R.\mathsf{rate}$ **or** $R.\mathsf{rate}$ = '_') **and not exist** (
select * from N
where $N.\mathsf{cid} = R.\mathsf{cid}$ **and** $N.\mathsf{pos}$ = 'RHS' **and** $N.\mathsf{att}$ = 'rate' **and** $R_2.\mathsf{rate}$ =N.val))

The SQL queries generated for error detection can be simplified as follows. As shown in Example 10, when checking patterns imposed by enc, enc_L or enc_R, the queries need not consider attributes A if $t[A]$ is 'null' for each tuple t in the table. Similarly, if an attribute A does not appear in any tuple in $\mathsf{enc}_{\neq}$, the queries need not check A either. From this, it follows that we do not even need to generate those attributes with only 'null' patterns for data tables enc, enc_L or enc_R when encoding CIND^{p}s or CFD^{p}s. □

6 Related Work

Constraint-based data cleaning was introduced in [2], which proposed to use dependencies, *e.g.,* FDs, INDs and denial constraints, to detect and repair errors

in real-life data (see, *e.g.,* [11] for a comprehensive survey). As an extension of traditional FDs, CFDs were developed in [14], for improving the quality of data. It was shown in [14] that the satisfiability and implication problems for CFDs are NP-complete and coNP-complete, respectively. Along the same lines, CINDs were proposed in [8] to extend INDs. It was shown [8] that the satisfiability and implication problems for CINDs are in constant time and EXPTIME-complete, respectively. SQL techniques were developed in [14] to detect errors by using CFDs, but have not been studied for CINDs. This work extends the static analyses of conditional dependencies of [14, 8], and has established several new complexity results, notably in the absence of finite-domain attributes (*e.g.,* Theorems 2, 6, 8). In addition, it is the first work to develop SQL-based techniques for checking violations of CINDs and violations of CFD^{p}s and CIND^{p}s taken together.

Extensions of CFDs have been proposed to support disjunction and negation [7], cardinality constraints and synonym rules [9], and to specify patterns in terms of value ranges [18]. While CFD^{p}s are more powerful than the extension of [18], they cannot express disjunctions [7], cardinality constraints and synonym rules [9]. To our knowledge no extensions of CINDs have been studied. This work is the first full treatment of extensions of CFDs and CINDs by incorporating built-in predicates ($\neq, <, \leq, >, \geq$), from static analyses to error detection.

Methods have been developed for discovering CFDs [10, 18, 15] and for repairing data based on either CFDs [13], traditional FDs and INDs taken together [5], denial constraints [4, 12], or aggregate constraints [16]. We defer the treatment of these topics for CFD^{p}s and CIND^{p}s to future work.

A variety of extensions of FDs and INDs have been studied for specifying constraint databases and constraint logic programs [3, 6, 19, 20]. While the languages of [3, 19] cannot express CFDs, constraint-generating dependencies (CGDs) of [3] and constrained tuple-generating dependencies (CTGDs) of [20] can express CFD^{p}s, and CTGDs can also express CIND^{p}s. The increased expressive power of CTGDs comes at the price of a higher complexity: both their satisfiability and implication problems are undecidable. Built-in predicates and arbitrary constraints are supported by CGDs, for which it is not clear whether effective SQL queries can be developed to detect errors. It is worth mentioning that Theorems 2 and 6 of this work provide lower bounds for the consistency and implication analyses of CGDs, by using patterns with built-in predicates only.

7 Conclusions

We have proposed CFD^{p}s and CIND^{p}s, which further extend CFDs and CINDs, respectively, by allowing patterns on data values to be expressed in terms of $\neq, <, \leq, >$ and $\geq$ predicates. We have shown that CFD^{p}s and CIND^{p}s are more powerful than CFDs and CINDs for detecting errors in real-life data. In addition, the satisfiability and implication problems for CFD^{p}s and CIND^{p}s have the same complexity bounds as their counterparts for CFDs and CINDs, respectively. We have also provided automated methods to generate SQL queries for detecting errors based on CFD^{p}s and CIND^{p}s. These provide commercial DBMS with an immediate capability to capture errors commonly found in real-world data.

One topic for future work is to develop a dependency language that is capable of expressing various extensions of CFDs (*e.g.,*CFD^{p}s, eCFDs [7] and CFD^{c}s [9]), without increasing the complexity of static analyses. Second, we are developing effective algorithms for discovering CFD^{p}s and CIND^{p}s, along the same lines as [10, 18, 15]. Third, we plan to extend the methods of [5, 13] to repair data based on CFD^{p}s and CIND^{p}s, instead of using CFDs [13], traditional FDs and INDs [5], denial constraints [4, 12], and aggregate constraints [16].

Acknowledgments. Fan and Ma are supported in part by EPSRC E029213/1. Fan is a Yangtze River Scholar at Harbin Institute of Technology.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, 1999.
3. M. Baudinet, J. Chomicki, and P. Wolper. Constraint-Generating Dependencies. *J. Comput. Syst. Sci.*, 59(1):94–115, 1999.
4. L. E. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Inf. Syst.*, 33(4-5):407–434, 2008.
5. P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, 2005.
6. P. D. Bra and J. Paredaens. Conditional dependencies for horizontal decompositions. In *ICALP*, 1983.
7. L. Bravo, W. Fan, F. Geerts, and S. Ma. Increasing the expressivity of conditional functional dependencies without extra complexity. In *ICDE*, 2008.
8. L. Bravo, W. Fan, and S. Ma. Extending dependencies with conditions. In *VLDB*, 2007.
9. W. Chen, W. Fan, and S. Ma. Incorporating cardinality constraints and synonym rules into conditional functional dependencies. *IPL*, 109(14):783–789, 2009.
10. F. Chiang and R. J. Miller. Discovering data quality rules. In *VLDB*, 2008.
11. J. Chomicki. Consistent query answering: Five easy pieces. In *ICDT*, 2007.
12. J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.
13. G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, 2007.
14. W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(2), 2008.
15. W. Fan, F. Geerts, L. V. Lakshmanan, and M. Xiong. Discovering conditional functional dependencies. In *ICDE*, 2009.
16. S. Flesca, F. Furfaro, and F. Parisi. Consistent query answers on numerical databases under aggregate constraints. In *DBPL*, 2005.
17. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
18. L. Golab, H. J. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. In *VLDB*, 2008.
19. M. J. Maher. Constrained dependencies. *TCS*, 173(1):113–149, 1997.
20. M. J. Maher and D. Srivastava. Chasing Constrained Tuple-Generating Dependencies. In *PODS*, 1996.
21. R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *JCSS*, 54(1), 1997.