Extending Conditional Dependencies with Built-in Predicates

Shuai Ma, Liang Duan, Wenfei Fan, Chunming Hu, and Wenguang Chen

Abstract—This paper proposes a natural extension of conditional functional dependencies (CFDs [22]) and conditional inclusion dependencies (CINDs [30]), denoted by \( \text{CFD}^p \) and \( \text{CIND}^p \), respectively, by specifying patterns of data values with \( \neq, <, \leq, >, \) and \( \geq \) predicates. As data quality rules, \( \text{CFD}^p \) and \( \text{CIND}^p \) are able to capture errors that commonly arise in practice but cannot be detected by CFDs and CINDs. We establish two sets of criteria for central technical problems associated with \( \text{CFD}^p \) and \( \text{CIND}^p \). (a) One concerns the satisfiability and implication problems for \( \text{CFD}^p \) and \( \text{CIND}^p \), 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 \( \text{CFD}^p \) and \( \text{CIND}^p \) retain the same complexity as their CFDs and CINDs counterparts. (b) The other concerns validation of \( \text{CFD}^p \) and \( \text{CIND}^p \). We show that given a set \( \Sigma \) of \( \text{CFD}^p \) and \( \text{CIND}^p \) 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 \( \Sigma \). We also experimentally verified the efficiency and effectiveness of our SQL based error detection techniques, using real-life data. This provides commercial DBMS with an immediate capability to detect errors based on \( \text{CFD}^p \) and \( \text{CIND}^p \).

Index Terms—Conditional dependencies, built-in predicates, functional dependencies, inclusion dependencies, data quality

1 INTRODUCTION

Extensions of traditional functional dependencies (FDs) and inclusion dependencies (INDs), known as conditional functional dependencies (CFDs [22]) and conditional inclusion dependencies (CINDs [30]), 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. It has been shown that conditional dependencies are able to capture more inconsistencies than FDs and INDs [17], [21], [30].

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

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:

\[
\begin{align*}
\text{item} & : (\text{id}; \text{name}; \text{type}; \text{price}; \text{float}; \\
\text{shipping}; \text{float}; \text{sale}; \text{bool}; \text{state}; \text{string}) \\
\text{tax} & : (\text{state}; \text{string}; \text{rate}; \text{float})
\end{align*}
\]

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., [3]) and conditional dependencies (CFDs, CINDs [22], [30]) on the data include the following:

\[
\begin{align*}
\text{cfd}_1 & : \text{item} (\text{id}; \text{name}; \text{type}; \text{price}; \text{sale}) \\
\text{cfd}_2 & : \text{tax} (\text{state}; \text{rate}) \\
\text{cfd}_3 & : \text{item} (\text{sale} = 'T' \rightarrow \text{shipping} = 0)
\end{align*}
\]

These are CFDs: (a) \( \text{cfd}_1 \) assures that the id of an item uniquely determines its name, type, price, shipping and sale; (b) \( \text{cfd}_2 \) states that state is a key for tax, i.e., for each state there is a unique sale tax rate; and (c) \( \text{cfd}_3 \) ensures that for any item tuple \( t \), if \( t[\text{sale}] = 'T' \) then \( t[\text{shipping}] \) must be 0; i.e., free shipping is provided for items on sale. Here \( \text{cfd}_3 \) is specified in terms of patterns of semantically related data values, namely, \( \text{sale} = 'T' \) and \( \text{shipping} = 0 \). It is to hold only on item tuples that match the pattern \( \text{sale} = 'T' \). In contrast, \( \text{cfd}_1 \) and \( \text{cfd}_2 \) 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 \( \text{cfd}_1, \text{cfd}_2 \) and \( \text{cfd}_3 \). 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 [22] or CINDs of [30], but can be expressed as the following dependencies:
(2) We establish the complexity bounds for the satisfiability and implication problems for CFPs and CINDs, taken separately or together (Section 4). The satisfiability problem is to determine whether a set \( \Sigma \) of dependencies has a nonempty model, i.e., whether the rules in \( \Sigma \) are consistent themselves. The implication problem is to decide whether a set \( \Sigma \) of dependencies entails another dependency \( \varphi \), i.e., whether the rule \( \varphi \) is redundant in the presence of the rules in \( \Sigma \). These are the central technical problems associated with any dependency language.

We show that despite the increased expressive power, CFPs and CINDs do not increase the complexity for reasoning about them. In particular, we show that the satisfiability and implication problems remain (a) \( \text{NP} \)-complete and \( \text{coNP} \)-complete for CFPs, respectively, (b) in \( \text{O}(1) \)-time (constant-time) and \( \text{EXPTIME} \)-complete for CINDs, respectively, and (c) are undecidable when CFPs and CINDs are taken together. These are the same as their CFDS and CINDs counterparts [30]. In contrast, data with linearly ordered domains often makes our lives harder [35].

(3) We provide SQL-based techniques to detect errors based on CFPs and CINDs (Section 5). Given a set \( \Sigma \) of CFPs and CINDs 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 \( \Sigma \). Further, the SQL queries are independent of the size and cardinality of \( \Sigma \). These provide the capability of detecting errors in a single relation (CFDp) and across different relations (CINDp) within the immediate reach of commercial DBMS.

(4) Using real-life data (HOSP and DBLP), we finally conduct an extensive experimental study (Section 6). We show that (a) the running time of CFPs and CINDs is comparable to their CFDS and CINDs counterparts, which is consistent with the static analyses in Section 4, and (b) CFDPs and CINDPs are able to capture more errors than their CFDS and CINDs counterparts (22% on HOSP and 75% on DBLP), due to the increased expressive power.

Related work. This paper is an extension of our earlier work [12] by adding (a) the proofs for the complexity bounds for the satisfiability and implication analyses of CFPs and CINDs, separately and taken together (Section 4), and (b) an extensive experimental study of CFPs and CINDs (Section 6), which was not investigated in [12].

Recently, data dependencies have generated renewed interests for improving data quality [5], [10], [14], [15], [22], [28], [30], [33], [36]. Constraint-based data cleaning was introduced in [4], which proposed to use dependencies, e.g., FDs, INDS and denial constraints, to detect and repair errors in real-life data (see, e.g., [3], [15], [33] for...
details). Data dependencies have been studied for relational databases since the introduction of FDS by Codd [16] in 1972 (see, e.g., [3] for details), and the theory of INDS was established in [11], which developed a sound and complete inference system and the PSPACE-completeness for the implication analysis of INDS. As an extension of traditional FDS, CFDs were developed in [22], for improving the quality of data. It was shown in [22] that the satisfiability and implication problems for CFDs are NP-complete and EXPTIME-complete, respectively. Along the same lines, CINDs [30] were proposed to extend INDS, and it was shown [30] that the satisfiability and implication problems for CINDs are in constant time and EXPTIME-complete, respectively. SQL techniques were developed in [22] to detect errors by using CFDs, but have not been studied for CINDs. This work extends the static analyses of conditional dependencies of [22], [30], and has established several new complexity results, notably in the absence of finite-domain attributes (e.g., Theorems 2, 8 and Proposition 6). In addition, it is the first work to develop SQL techniques for checking violations of CINDs and violations of CFDs and CINDs taken together.

Extensions of CFDs have been proposed to support disjunction and negation [10], cardinality constraints and synonym rules [13], and to specify patterns in terms of value ranges [28]. While CFDs are more powerful than the extension of [28], they cannot express disjunctions [10], cardinality constraints and synonym rules [13]. 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 [14], [28], CFDs [36] and CINDs [5] and for repairing data based on either CFDs [17], traditional FDS and INDS taken together [8], CFDs and CINDs taken together [19], denial constraints [7], aggregate constraints [25], matching dependencies [20], matching dependencies and CFDs [24], or editing rules and master data [23]. We defer the treatment of these topics for CFDs and CINDs to future work.

A variety of extensions of FDS and INDS have been studied for specifying constraint databases and constraint logic programs [6], [9], [27], [31], [32]. While the languages of [6], [27], [31] cannot express CFDs, constraint-generating dependencies (CGDs) of [6] and constrained tuple-generating dependencies (CTGDs) of [32] can express CFDs, and CTGDs can also express CINDs. 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.

Observe that constraints specifying semantics with orderings have long been recognized, such as order dependencies [27] supporting the comparison of attributes with $=, <, \leq, >, \geq$, matching dependencies [20] and differential dependencies [34] that support the comparison of attributes with $=, \neq, <, \leq, >, \geq$ for record matching. However, different from CFDs and CFDs, these constraints do not specify conditions on those tuples such that the embedded FDS hold. Further, it is also possible that other existing constraints could be improved by incorporating these built-in predicates, such as metric functional dependencies [29].

### 2 Extending CFDs with Predicates

We now define conditional functional dependencies with predicates, denoted by CFDs, by extending CFDs and CINDs with built-in predicates ($\neq, <, \leq, >, \geq$) in addition to equality ($\equiv$).

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

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

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 we separate the X and Y attributes in a pattern tuple with '. We simply write $\varphi$ as $(X \rightarrow Y, T_p)$ when $\varphi$ is clear from the context, and denote X as $\text{LHS}(\varphi)$ and Y as $\text{RHS}(\varphi)$, respectively.

**Example 2:** The dependencies $\text{cfd}_1\text{cfd}_2$ and $\text{pfd}_1\text{pfd}_4$ that we have seen in Example 1 can all be expressed as CFDs. Some of these CFDs are illustrated in Fig. 2, in which $\varphi_1$ is for FD $\text{cfd}_2$, $\varphi_2$ is for FD $\text{cfd}_3$, $\varphi_3$ is for $\text{pfd}_2$, and $\varphi_4$ is for $\text{pfd}_4$, respectively.

**Semantics.** Consider CFD $\varphi = R(X \rightarrow Y, T_p)$, where $T_p = \{t_{p1}, \ldots, t_{pn}\}$.

A data tuple $t$ of $R$ is said to match $LHS(\varphi)$, denoted by $t[X] = T_p[X]$, if for each tuple $t_{pi}$ ($i \in [1, k]$) in $T_p$ and each

$$
\begin{align*}
(1) & \; \varphi_1 = \text{tax (state} \rightarrow \text{rate), } T_1 = (T_1: \begin{array}{l} \text{state|rate} \\
\end{array}) \\
(2) & \; \varphi_2 = \text{item (sale} \rightarrow \text{shipping), } T_2 = (T_2: \begin{array}{l} \text{sale|shipping} \\
\end{array}) \\
(3) & \; \varphi_3 = \text{item (sale, price} \rightarrow \text{shipping), } T_3 = (T_3: \begin{array}{l} \text{sale|price|shipping} \\
\end{array}) \\
(4) & \; \text{CFD} \varphi_4 = \text{item (sale} \rightarrow \text{price), } T_4 = (T_4: \begin{array}{l} \text{sale|price} \\
\end{array})
\end{align*}

Figure 2. Example CFDs

$\phi_1 = \text{tax (state} \rightarrow \text{rate), } T_1 = (T_1: \begin{array}{l} \text{state|rate} \\
\end{array})$
attribute $A$ in $X$, either (a) $t_{p_1}[A]$ is the wildcard ‘\_’ (which matches any value in $\text{dom}(A)$), or (b) $t[A]$ op $a$ if $t_{p_1}[A]$ is ‘op $a’$, where the operator op ($=, \neq, <, \leq, >, \geq$) is interpreted by its standard semantics. Similarly, the notion that $t$ matches RHS($\phi$) is defined, denoted by $t[Y] \models \text{RHS}(\phi)$.

Intuitively, each pattern tuple $t_{p_i}$ ($i \in [1,k]$) specifies a condition via $t_{p_i}[X]$, and $t[X] \models \text{RHS}(\phi)$ if $t[X]$ satisfies the conjunction of all these conditions. Similarly, $t[Y] \models \text{RHS}(\phi)$ if $t[Y]$ matches all the patterns specified by $t_{p_i}[Y]$ for all pattern tuples $t_{p_i}$ in $T_p$.

An instance $I$ of $R$ satisfies the CFD$^p$ $\phi$, denoted by $I \models \phi$, if for each pair of tuples $t_1, t_2$ in $I$, if $t_1[X] = t_2[X] \models \text{RHS}(\phi)$, then $t_1[Y] = t_2[Y] \models \text{LHS}(\phi)$. 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 $\phi$ is imposed only on the subset of tuples in $I$ that match LHS($\phi$), 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) $t_1[Y] \models \text{RHS}(\phi)$, which assures that the constants in $t_1[Y]$ match the constants in $t_2[Y]$ for all $t_{p_i}$ in $T_p$. Note that here tuples $t_1$ and $t_2$ can be the same.

An instance $I$ of $R$ satisfies a set $\Sigma$ of CFD$^p$'s, denoted by $I \models \Sigma$, if $I \models \phi$ for each CFD$^p$ $\phi$ in $\Sigma$.

**Example 3:** The instance $D_0$ of Fig. 1 satisfies $\phi_1$ and $\phi_2$ of Fig. 2, but neither $\phi_3$ nor $\phi_4$. Indeed, tuple $t_3$ violates ($i.e.$, does not satisfy) $\phi_3$, since $t_3[\text{sale}] = \text{S}'$ and $t_3[\text{price}] \leq 40$, but $t_3[\text{shipping}]$ is 20 instead of 6. Note that $t_4$ matches LHS($\phi_3$) since it satisfies the condition specified by the conjunction of the pattern tuples in $T_2$. Similarly, $t_1$ violates $\phi_4$, since $t_1[\text{sale}] = \text{T}'$ but $t_1[\text{price}] > 9.99$. 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 $X \rightarrow Y$ [3] can be expressed as a CFD$^p$ ($X \rightarrow Y$, $T_p$) in which $T_p$ contains a single tuple consisting of ‘\_’ only, without constants. (2) A CFD ($X \rightarrow Y$, $T_p$) [22] with $T_p = \{t_1, t_2, \ldots, t_k\}$ can be expressed as a set $\{\phi_1, \ldots, \phi_k\}$ of CFD$^p$'s such that for each $i \in [1,k]$, $\phi_i = (X \rightarrow Y, T_p)$, where $T_p$ contains the pattern tuple $t_i$ of $T_p$, only defined with equality (=) only. For example, $\phi_1$ and $\phi_2$ in Fig. 2 are CFD$^p$'s representing FD cfd3 and CFD cfd3 in Example 1, respectively. Note that all data quality rules in [14, 28] can be expressed as CFD$^p$'s.

### 3 Extending CINDs with Predicates

Similar to CFD$^p$'s, we define conditional inclusion dependencies with predicates, denoted by CIND$^p$'s, by extending CINDs [30] with built-in predicates ($\neq, <, \leq, >, \geq$) in addition to equality (=). Consider two relational schemas $R_1$ and $R_2$.

**Syntax.** A CIND$^p$ $\psi$ is a pair $(R_1[X], X_{D} \subseteq R_2[Y], T_p)$, where (1) $X, X_{D}$ and $Y, Y_{D}$ are lists of attributes in attr$(R_1)$ and attr$(R_2)$, respectively; (2) $R_1[X] \subseteq R_2[Y]$ is a standard IND, referred to as the IND embedded in $\psi$; and (3) $T_p$ is a tableau, called the pattern tableau of $\psi$ defined over attributes $X_{D} \cup Y_{D}$, and for each $A$ in $X_{D}$ or $Y_{D}$, each pattern tuple $t_{p} \in T_p$, $t_{p}[A]$ is an either an unnamed variable ‘\_’ that draws values from $\text{dom}(A)$, or ‘op+a’, where op is one of $=, \neq, <, \leq, >, \geq$ and ‘a’ is a constant in dom$(A)$.

We denote $X \cup X_{p}$ as LHS($\psi$), $Y \cup Y_{p}$ as RHS($\psi$), and separate the $X_{p}$ and $Y_{p}$ attributes in a pattern tuple with ‘\_’. We also use nil to denote an empty list.

**Example 4:** Figure 3 shows two example CIND$^p$'s: $\psi_1$ expresses the pind$_1$ in Example 1, and $\psi_2$ refines $\psi_1$ by stating that for any item tuple $t_i$, 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., $\psi_2$ assures that the sale tax rate in Delaware is 0.

**Semantics.** Consider CIND$^p$ $\psi = (R_1[X] \subseteq R_2[Y], T_p)$. An instance $(I_1, I_2)$ of $(R_1, R_2)$ satisfies the CIND$^p$ $\psi$, denoted by $(I_1, I_2) \models \psi$ if for each tuple $t_1 \in I_1$, if $t_1[X] \models \text{RHS}(\psi)$, then there exists a tuple $t_2 \in I_2$ such that $t_1[X] = t_2[Y]$ and $t_2[Y_{D}] \models T_p[Y_{D}]$.

That is, if $t_1[X]$ matches the pattern tableau $T_p[X]$, then $\psi$ assures the existence of $t_2$ such that $(1) t_1[Y] = t_2[Y]$ as needed by the standard IND embedded in $\psi$, and moreover, (2) $t_2[Y_{D}]$ must match the pattern tableau $T_p[Y_{D}]$. In other words, $\psi$ is “conditional” since its embedded IND is applied only to the subset of tuples in $I_1$ that match $T_p[X]$, and $T_p[Y_{D}]$ 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 all the pattern tuples in $T_p$.

**Example 5:** The instance $D_0$ of item and tax in Fig. 1 violates CIND$^p$ $\psi_1$. Indeed, tuple $t_1$ in item matches LHS($\psi_1$) since $t_1[\text{type}] \neq \text{‘art’}$, but there is no tuple in tax such that $t[\text{state}] = t_1[\text{state}] = \text{‘WA’}$. In contrast, $D_0$ satisfies $\psi_2$.

We say that a database $D$ satisfies a set $\Sigma$ of CIND$^p$'s, denoted by $D \models \Sigma$, if $D \models \psi$ for each $\psi \in \Sigma$.

**Safe CIND$^p$'s.** We say a CIND$^p$ $(R_1[X] \subseteq R_2[Y], T_p)$ is unsafe if there exist pattern tuples $t_{p_1}, t_{p_2}$ in $T_p$ such that either (a) there exists $B \in Y_{D}$ such that $t_{p_1}[B]$ and $t_{p_2}[B]$ are not satisfiable when taken together, or (b) there exist $C \in Y$, $A \in X$ such that $A$ corresponds to $C$ in the embedded IND and $t_{p_1}[C]$ and $t_{p_2}[A]$ are not satisfiable when taken together; e.g., $t[\text{price}] = 9.99$ and $t'[\text{price}] \geq 19.99$.

Obviously unsafe CIND$^p$'s do not make sense: no nonempty databases satisfy 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.** (1) A standard IND $(R_1[X] \subseteq R_2[Y])$ can be expressed as a CIND$^p$ $(R_1[X], \text{nil} \subseteq R_2[Y], T_p)$ such that $T_p$ is simply a empty set. (2) A CIND $(R_1[X] \subseteq R_2[Y], T_p)$ with $T_p = \{t_1, t_2, \ldots, t_k\}$ can be expressed as a set $\{\psi_1, \ldots, \psi_k\}$ of CIND$^p$'s, where for each $i \in [1,k]$, $\psi_i = (R_1[X], X_{D} \subseteq R_2[Y], T_{p,i})$ such that $T_{p,i}$ consists of the pattern tuple $t_{p,i}$ of $T_{p,i}$ defined with equality (=) only.

### 4 Reasoning about CFD$^p$'s and CIND$^p$'s

The satisfiability and implication problems are the two classical questions associated with any dependency languages [3, 22, 30]. In this section we investigate these problems for CFD$^p$'s and CIND$^p$'s, separately and taken together.

#### 4.1 Satisfiability Analyses

The satisfiability problem is to determine, given a set $\Sigma$ of constraints, whether there exists a nonempty database that satisfies $\Sigma$.
The satisfiability analysis of conditional dependencies is not only of theoretical interest, but is also important in practice. Indeed, when CFD's and CINDP'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 [5], [14], [28].

**Satisfiability analysis of CFD's.** Given any FDS, one does not need to worry about their satisfiability as any set of FDS is always satisfiable. However, as observed in [22], for a set $\Sigma$ 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's, the same problem exists when it comes to CFD's.

**Example 6:** Consider a CFD $\varphi = (R : A \rightarrow B, T_p)$ such that $T_p = \{(\_ | = a), (\_ | \neq a)\}$. There is no nonempty instance $I$ of $R$ that satisfies $\varphi$. Indeed, for any $R$ tuple $t$, $\varphi$ requires that both $t[B] = a$ and $t[B] \neq a$, which is impossible. □

This problem is already NP-complete for CFDs [22]. Below we show that it remains the same complexity for CFD's despite their increased expressive power.

**Proposition 1:** The satisfiability problem for CFD's is NP-complete.

**Proof:** The lower bound follows from the NP-hardness of their CFD counterparts [22], since CFDs are a special case of CFD's. The upper bound is verified by presenting an NP algorithm that, given a set $\Sigma$ of CFD's defined on a relational schema $R$, determines whether $\Sigma$ is satisfiable.

We next present an NP algorithm that, given a set $\Sigma$ of CFD's defined on a relational schema $R$, determines whether $\Sigma$ is satisfiable or not. The satisfiability problem has the following small model property: If there is a nonempty instance $I$ such that $I \models \Sigma$, then for any tuple $t \in I$, instance $I_t = \{t\}$ satisfies $\Sigma$. Thus it suffices to consider single-tuple instances $I = \{t\}$ for deciding whether $\Sigma$ is satisfiable.

Assume w.l.o.g. that the attributes $\text{attr}(R) = \{A_1, \ldots, A_m\}$ and the total number of pattern tuples in all pattern tableaux $T_p$ of CFD's in $\Sigma$ is $h$. For each $i \in [1, m]$, define the active domain of $A_i$ to be a set $\text{dom}(A_i) = C_0 \cup C_1$, where (1) $C_0$ consists of all constants in $T_p[A_i]$ of all pattern tableaux $T_p$ in $\Sigma$, and if $C_0$ is empty, we further let $C_0 = \{a_1, a_2\}$, where $a_1, a_2 \in \text{dom}(A_i)$ and $a_1 \neq a_2$, and (2) $C_1$ contains the set of constants for the attributes whose domains have total orders, i.e., involved with predicates $\neq, <, \leq, >$ or $\geq$.

1. Arrange all constants in $C_0$ in the increasing order, and assume the resulting $C_0 = \{a_1, \ldots, a_k\}$ ($k \geq 1$).
2. Add a constant $b_{k+1} \notin \text{dom}(A_i)$ to $C_1$ such that $b_{k+1} < a_1$ if there exists one; And also add another constant $b_{k+2} \notin \text{dom}(A_i)$ to $C_1$ such that $b_{k+2} < a_1$ and $b_{k+2} \neq b_{k+1}$ if there exists one;
3. Similarly, for each $j \in [1, k - 1]$, add a constant $b_{j+1} \in \text{dom}(A_i)$ to $C_1$ such that $a_j < b_{j+1} < a_{j+1}$ if there exists one; And also add another constant $b_{j+2} \in \text{dom}(A_i)$ to $C_1$ such that $a_j < b_{j+2} < a_{j+1}$ and $b_{j+2} \neq b_{j+1}$ if there exists one;
4. Finally, add a constant $b_{k+1} \in \text{dom}(A_i)$ to $C_1$ such that $b_{k+1} > a_k$ if there exists one; And also add another constant $b_{k+2} \in \text{dom}(A_i)$ to $C_1$ such that $b_{k+2} > a_k$ and $b_{k+2} \neq b_{k+1}$ if there exists one.

Moveover, the number of elements in $\text{dom}(A_i)$ is at most $3 \times h + 2$. Then one can easily verify that $\Sigma$ is satisfiable if there exists a mapping $\rho$ from $t[A_i]$ to $\text{dom}(A_i)$ ($i \in [1, m]$) such that $I = \{(\rho(t[A_i]), \ldots, \rho(t[A_m]))\}$ and $I \models \Sigma$.

We now give an NP algorithm as follows: (1) Guess an instance, which contains a single tuple $t$ of $R$ such that $t[A_i] \in \text{dom}(A_i)$ for each $i \in [1, m]$. (2) Check whether $I \models \Sigma$. If so the algorithm returns ‘yes’, and otherwise it repeats steps (1) and (2). Obviously step (2) can be done in PTIME in the size of $\Sigma$. Hence the algorithm is in NP, and so is the problem. □

It is known [22] that the satisfiability problem for CFD's 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's. This tells us that the increased expressive power of CFD'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 [22], the result below is new.

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

**Proof:** The problem is in NP by Proposition 1. Its NP-hardness is shown by reduction from the 3SAT problem, which is NP-complete (cf. [26]).

We next show the reduction from the 3SAT problem. Consider an instance $\phi = C_1 \land \cdots \land C_p$ of 3SAT, where all the variables in $\phi$ are $x_1, \ldots, x_m, C_j$ is of the form $y_{j_1} \lor y_{j_2} \lor y_{j_3}$ such that for each $i \in [1, 3], y_j$ is either $x_{p_i}$ or $\overline{x_{p_i}}$ for $p_j \in [1, m]$. Given $\phi$, we construct a relational schema $R$ and a set $\Sigma$ of CFD's defined on $R$ such that $\phi$ is satisfiable if $\Sigma$ is satisfiable.

1. We first define the relational schema $R(\bar{x}_1, \ldots, \bar{x}_m, C_1, \ldots, C_n, Z)$, where all attributes share a common infinite domain $\text{dom}$ that contains constant $a$. Intuitively, for each $R$ tuple $t$, $t[\bar{x}_1, \ldots, \bar{x}_m]$ specifies a truth assignement $\varepsilon$ for variables $x_1, \ldots, x_m$ of $\phi$, and $t[C_1]$ ($i \in [1, n]$) and $t[Z]$ are the truth values of clause $C_i$ and sentence $\phi$ w.r.t. the assignment $\varepsilon$, respectively.
2. We then construct the set CFD's $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \ldots \cup \Sigma_n \cup \Sigma_{n+1} \cup \Sigma_{n+2}$, defined as follows.
   a) $\Sigma_0$ contains $n+1$ CFD's, which intuitively encode the relationships of the truth values between the clauses
Given a set $\mathcal{C}$, for each clause $C_i (i \in [1, n])$, we add to $\Sigma_0$ a CFD $\varphi_{i} = (C_1, \ldots, C_n \rightarrow Z, T_{pi})$, in which $T_{pi} = \{ t_{pi} \}$ such that $t_{pi}[C_i, Z] = (\neq a \ || a)$ and $t_{pi}[C_i] = t_{pi}$ for any $j \neq i$ and $j \in [1, n]$. We also add to $\Sigma_0$ a CFD $\varphi_0 = (C_1, \ldots, C_n \rightarrow Z, T_p)$, where $T_p = \{(a, \ldots, a \ || a)\}$. Intuitively, we use $\neq a$ and $= a$ to represent false and true, respectively.

(b) For $i \in [1, n]$, $\Sigma_i$ contains 8 CFDs, which intuitively encode the relationships of the truth values between the clause $C_i$ and its three variables.

For clause $C_i = x_j \lor \exists x_j \lor x_j$, we define CFDs $\varphi_{i,1} = (X_j, X_j, X_j \rightarrow C_i, T_{pi,0})$ with $T_{pi,0} = \{(a, a < a, < a || a)\}$, (b) $\varphi_{i,2} = (X_j, X_j, X_j \rightarrow C_i, T_{pi,2})$ with $T_{pi,2} = \{(a, a \geq a, || a)\}$, and (c) $\varphi_{i,3} = (X_j, X_j, X_j \rightarrow C_i, T_{pi,3})$ with $T_{pi,3} = \{(a, a \geq a, || a)\}$. Similarly, we can define the rest 5 CFDs $\varphi_{i,4}, \varphi_{i,5}, \varphi_{i,6}$ and $\varphi_{i,7}$. Theorem 2, we can use $\neq a$ and $= a$ to represent false and true for a variable, respectively.

(c) $\Sigma_{n+1}$ contains a single CFD $\varphi_{n+1} = (Z \rightarrow Z, T_{p(n+1)})$ with $T_{p(n+1)} = \{(|| a)\}$. Intuitively, $\varphi_{n+1}$ requires that for any tuple $t$, $t[Z] = a$.

Observe that $\Sigma$ contains 8 variables for each CFD $\varphi$ in total. Thus the reduction is in PTIME.

We now show that $\phi$ is satisfiable iff $\Sigma$ is satisfiable.

Assume first that $\Sigma$ is satisfiable, then we show that there exists a nonempty instance $I$ of $R$ such that $I \models \Sigma$. Any tuple $t \in I$ forces $t[Z] = a$, (b) $\Sigma_0$ forces $t[C, Z] = \{a, a, a\}$, and (c) for each clause $C_i \in [1, n]$ with variables $x_j, x_j, x_j$, $\Sigma_i$ forces that $t[X_j, X_j, X_j]$ does not match the LHS of the CFDs that force $t[C_i] \neq a$. From the tuple $t$, we can construct a truth assignment $\xi$ of $\phi$ such that $\xi(x_i) = false$ if $t[X_i] < a$ and $\xi(x_i) = true$ if $t[X_i] > a$ (i $\in [1, m]$). Since $\{\} \models \Sigma$, it is easy to verify that the truth assignment $\xi$ makes $\phi$ true.

Conversely, if $\phi$ is satisfiable, there exists a truth assignment $\xi$ that makes $\phi$ true. We construct a tuple $t$ of $R$ as follows: (a) $t[C_1, \ldots, C_n, Z] = \{a, \ldots, a\}$ and (b) for each $i \in [1, m]$, $t[X_i] = a_i$ such that (a) $a_i \in dom$ and $a_i \geq a$ if $\xi(x_i) = true$ and $a_i < a$ otherwise. Let $I = \{t\}$, then one can easily verify that $I \models \Sigma$.

Putting this together, we have the conclusion.

Satisfiability analysis of CINDs. Like FDs, one can specify arbitrary INs and CINDs without worrying about their satisfiability. Below we show that CINDs preserve this nice property, by extending the proof of their counterpart in [30].

**Proposition 3**: Any set of CINDs is always satisfiable.

**Proof**: Given a set $\Sigma$ of CINDs over a database schema $\mathcal{R}(R_1, \ldots, R_n)$, we show that one can always construct a nonempty instance $D$ of $\mathcal{R}$ such that $D \models \Sigma$.

We build $D$ as follows. First, for each attribute $A$, define the active domain of $A$ to be a set $dom(A)$, which consists of certain data values in $dom(A)$. Second, using these active domains, we construct $D$.

(1) We start with the construction of active domains. (a) For each attribute $A$, initialize $dom(A)$ along the same lines as the one for CFDs in Proposition 1; (b) For each CIND $R(A_1, \ldots, A_n; X_p) \subseteq R_0[R_1, \ldots, R_n; Y_p]$, in $\Sigma$, let $dom(A_i) = dom(A_1) \cup dom(A_2)$ for each $i \in [1, m]$, and this rule is repeatedly applied until a fixpoint of $dom(A)$ is reached for all attributes $A$ in $\mathcal{R}$.

It is easy to verify that this process always terminates as we start with a finite set of data values.

(2) We next construct the database instance $D$. For each relation $R_1(A_1, \ldots, A_k) \in \mathcal{R}$, we define $I = dom(A_1) \times \cdots \times dom(A_k)$, where $\times$ is the Cartesian Product operation [3]. Let $D = \{I_1, \ldots, I_n\}$, then it is easy to verify that $D$ is nonempty and $D \models \Sigma$.\hfill $\Box$

**Satisfiability analysis of CFDs and CINDs.** The satisfiability problem for CFDs and CINDs taken together is undecidable [30]. Since CFDs and CINDs subsume FDs and CINDs, respectively, we immediately have the following.

**Corollary 4**: The satisfiability problem for CFDs and CINDs is undecidable.\hfill $\Box$

### 4.2 Implication Analyses

The implication problem is to determine, given a set $\Sigma$ of dependencies and another dependency $\phi$, whether or not $\Sigma$ entails $\phi$, 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 rules, and thus improve the performance of error detection and repairing based on the rules [22], [30].

**Example 7**: The CFDs in Fig. 2 imply another CFD $\varphi = (item, price \rightarrow shipping, T)$, where $T$ consists of a single pattern tuple $(sale = 'T', price = 30 \ | \ shipping = 6)$. Thus, in the presence of the CFDs in Fig. 2, $\varphi$ is redundant.\hfill $\Box$

**Implication analysis of CFDs.** We first show that the implication problem for CFDs retains the same complexity as their CFDs counterpart, verified by extending the proof of their counterpart in [22].

**Proposition 5**: The implication problem for CFDs is conp-complete.

**Proof**: The lower bound follows from the conp-hardness of their CFDs counterpart [22], since CFDs are a special case of CFDs. The conp upper bound is verified by presenting an NP algorithm for its complement problem for determining whether $\Sigma \not\models \phi$.

We next present the a NP algorithm for its complement problem. The algorithm is based on a small model property: if $\varphi = R(X \rightarrow Y, T_p)$ and $\Sigma \not\models \varphi$, then there exists an instance $I$ of $R$ with two tuples $t_1$ and $t_2$ such that $I \models \Sigma$ and $t_1[X] = t_2[X] \triangleright T_p[X]$, but either $t_1[Y] \neq t_2[Y]$ or $t_1[Y] \neq T_p[Y]$ (resp. $t_1[Y] \neq T_p[Y]$). Thus it suffices to consider instances $I$ with two tuples only for deciding whether $\Sigma \not\models \phi$.

Assume that the attributes $attr(R) = \{A_1, \ldots, A_n\}$. For each $i \in [1, m]$, let $adom(A_i)$ be the active domain defined in Proposition 1. Then one can easily verify that $\Sigma \not\models \varphi$ iff there exist two mappings $\varphi_1$ and $\varphi_2$ from all attributes $A_i$ to $adom(A_i)$ ($i \in [1, m]$) such that $I = \{(\varphi_1(A_1), \ldots, \varphi_1(A_m)), (\varphi_2(A_1), \ldots, \varphi_2(A_m))\}$, $I \models \Sigma$, but $I \not\models \varphi$.\hfill $\Box$
Based on these, we give an NP algorithm as follows: (1) Guess two $R$ tuples $t_1$ and $t_2$ such that $t_1[A_i], t_2[A_i] \in \text{adom}(A_i)$ for each $i \in [1, m]$. (2) Check whether $I = \{t_1, t_2\}$ satisfies $\Sigma$, but not $\phi$. If so the algorithm returns ‘yes’, and otherwise it repeats steps (1) and (2). Obviously step (2) can be done in $\text{PTIME}$ in the size of $\Sigma$. Hence the algorithm is in $\text{NP}$, and so is the problem.

Similar to the satisfiability analysis, it is known [22] that the implication analysis of CFDs is in $\text{PTIME}$ 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$\overline{\phi}$s, which does not find a counterpart in [22].

**Proposition 6**: In the absence of finite domain attributes, the implication problem for CFD$\overline{\phi}$s is $\text{conP}$-complete.

**Proof**: The problem is in $\text{conP}$ 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 $\Sigma \not\models \phi$.

We next show the reduction from the 3SAT problem to the complement problem of the implication problem for CFD$\overline{\phi}$s, where 3SAT is $\text{NP}$-complete (cf. Proposition 2). Given an instance $\phi$ of 3SAT, we construct a relational schema $R$ and a set $\Sigma \cup \{\phi\}$ of CFD$\overline{\phi}$s defined on $R$ such that $\phi$ is satisfiable iff $\Sigma \not\models \phi$.

The relational schema $R$ and the set $\Sigma$ of CFD$\overline{\phi}$s are the same as the corresponding ones in Proposition 2. Moreover, $\phi$ is defined as $(Z \rightarrow T, \psi)$, where $T = \{\neg | \phi \neq a\}$. Intuitively, $\phi$ requires that for any $R$ tuple $t$, $t[Z] \neq a$. Along the same lines as Proposition 2, one can easily verify that $\phi$ is satisfiable iff $\Sigma \not\models \phi$. Thus the problem is conP-hard.

**Implication analysis of CFD$\overline{\phi}$s**: We next show that CFD$\overline{\phi}$s do not make their implication analysis harder, verified by extending the proof of their CIND$\overline{\phi}$s counterpart given in [30].

**Proposition 7**: The implication problem for CFD$\overline{\phi}$s is EXPTIME-complete.

**Proof**: The implication problem for CIND$\overline{\phi}$s is EXPTIME-hard [30]. Since CIND$\overline{\phi}$s subsume CINDs, the lower bound carries over to CFD$\overline{\phi}$s immediately. The EXPTIME upper bound is shown by presenting an EXPTIME algorithm that, given a set $\Sigma \cup \{\psi\}$ of CFD$\overline{\phi}$s over a database schema $\mathcal{R}$, determines whether $\Sigma \models \psi$ or not.

We next present the EXPTIME algorithm. Consider $\mathcal{R} = (R_1, \ldots, R_n)$ and $\psi = (R_0[X_0; Y_0] \subseteq R_0(X_0, Y_0), T_0)$. And for each attribute $A$, define the active domain $\text{adom}(A)$ of $A$ based on $\Sigma \cup \{\psi\}$ along the same line as the proof of Proposition 3. One can easily verify that if $\Sigma \not\models \psi$, there exists a non-empty instance $D$ of $\mathcal{R}$ such that (a) $D \models \Sigma$ and $D \not\models \psi$, and (b) $D$ consists of data values from the active domains only.

The detailed EXPTIME algorithm is given as follows.

1. We first build a labeled directed graph $G(V, E, l)$. Each node $u \in V$ is a possible tuple ‘$R_i : t_i$’ such that $t_i[A] \in \text{adom}(A)$ for each attribute $A \in \text{attr}(R_i)$. There is an edge $e = (R_i, t_i') \rightarrow (R_j, t_j')$ in $E$ if there exists a CIND$\overline{\phi}$ $\phi = (R_0[U_0], R_0[V_0], \ldots, R_0[T_0])$ in $\Sigma$ such that $t_i[U_0] \times T_0 = t_i[U_0]$, $t_i[V] = t_j[U]$ and $t_i[V_0] \times T_0 = t_i[V_0]$, and $e$ is labeled with the CIND$\overline{\phi}$, i.e., $\phi \in \{e\}$. Note that an edge may have multiple labels.

2. Let $S_0$ be the set of nodes ‘$R_a : t_a$’ such that $t_a[X_0] \times T_0 = t_a[X_0]$, and $S_b$ be the set of nodes ‘$R_b : t_b$’ such that $t_b[Y_0] \times T_0 = t_b[Y_0]$, respectively.

3. For each node $u = ‘R_a : t_a$’ in $S_0$, let $G_u$ be the induced subgraph of $G$ that contains all the nodes reachable from $u$, and exactly the edges that appear in $G$ over the same set of nodes. We also refer to $u$ as the root of $G_u$.

4. For an induced subgraph $G_u$ of $G$ with root $u = ‘R_a : t_a$’, we derive another graph $G_u'$ by recursively removing edges as follows. For any $v$ in $G_u$, if $v$ has a child $v'$ from which no nodes in ‘$R_b : t_b$’ in $S_b$ with $t_b[X] = t_b[Y]$ are reachable, then for all children $v''$ of $v$, we remove from labels $l(v, v'')$ all the labels in $l(v', v'')$, and edge $(v, v'')$ is removed when $l(v, v'')$ becomes empty.

5. If there exists a subgraph $G_u'$ derived from an induced subgraph $G_u$ of $G$ with root $u = ‘R_a : t_a$’ such that no nodes ‘$R_b : t_b$’ in $S_b$ with $t_b[X] = t_b[Y]$ are reachable from $u$, we return ‘no’, and return ‘yes’, otherwise.

It can be verified that (a) if the algorithm returns ‘no’, we can construct an instance $D$ such that $D \models \Sigma$, but not $\psi$, by collecting those tuples attached on the end nodes of edges whose labels become empty at step 4; and (b) if the algorithm returns ‘yes’, there exist no instances $D$ such that $D \models \Sigma$, but not $\psi$.

We next show that the above algorithm indeed runs in exponential time: (a) The number of nodes in graph $G$ is bounded by the maximum number of tuples in a database instance on $\mathcal{R}$. Let $|\Sigma \cup \{\psi\}|$ be the size of $\Sigma$ and $\psi$, and $|\mathcal{R}|$ be the sum of arities of all relations in $\mathcal{R}$. Then the number of tuples in a database instance is bounded by $O(|\Sigma \cup \{\psi\}|^{\frac{|\mathcal{R}|}{3}})$; (b) The number of nodes in sets $S_0$ or $S_b$ is bounded by the maximum number of tuples in a database tree; (c) The induced subgraph and the reachability testing can be done in linear-time in the size of the input [18].

Putting all these together, we have shown that the algorithm runs in exponential time. And, hence, the problem is EXPTIME.

It is known [30] that the implication problem is $\text{PSPACE}$-complete for CINDs defined with infinite domain attributes. Similar to Theorem 6, below we show that this no longer holds for CFD$\overline{\phi}$s.

**Theorem 8**: In the absence of finite domain attributes, the implication problem for CFD$\overline{\phi}$s remains EXPTIME-complete.

**Proof**: The problem is in EXPTIME by 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, that is known to be EXPTIME-complete [30].

We next present the reduction from the implication problem for CINDs in the general setting. Given a set $\Sigma \cup \{\psi\}$ of CINDs defined on a database schema $\mathcal{R} = (R_1, \ldots, R_n)$, we construct another database schema $\mathcal{R}' = (R_1', \ldots, R_n')$, in which each relation $R_i'$ $(i \in [1, n])$ consists of infinite domain attributes only, and a set $\Sigma' \cup \{\psi\}$ of CIND$\overline{\phi}$s on $\mathcal{R}'$ such that $\Sigma \models \psi$ iff $\Sigma' \models \psi'$.
We start with constructing $R'$. For each $R_i(A_1, \ldots, A_k)$ of $\mathcal{R}$, we define $R'_i(A'_1, \ldots, A'_k)$ such that for each attribute $A'_j (j \in [1, k])$, let $\text{dom}(A'_j) = \text{dom}(A_j)$ if $\text{dom}(A_j)$ is infinite, and let $\text{dom}(A'_j)$ be integer, a totally ordered infinite domain, if $\text{dom}(A_j)$ is finite. Moreover, we define a mapping $\rho_{i,j}$ for each finite domain $\text{dom}(A_j) = \{a_1, \ldots, a_n\}$ to integer: (a) Randomly choose $h$ consecutive integers $\{b_1, \ldots, b_h\}$ such that for each $i \in [1, h - 1]$, $b_{i+1} = b_i + 1$. (b) We now define the mapping $\rho_{i,j}(a_i) = b_i$ for $i \in [1, h]$. Moreover, we require two extra integers $b_0 = b_1 - 1$ and $b_{h+1} = b_h + 1$, denoted as $\rho_{i,j}(a_0)$ and $\rho_{i,j}(a_{j+1})$. Note that this is always doable. For clarity, we also denote $\rho_{i,j}$ as $\rho$ when it is clear from the context.

We next define $\Sigma' \psi'$ on $R'$ based on the mappings defined above. For each CIND $\{R_n[X; A_1, \ldots, A_{m_n}] \subseteq R_n[Y; B_1, \ldots, B_{m_n}], T_p\}$ in $\Sigma$ and each $t_p \in T_p$, we define another CIND $\{R'_n[X'; A'_1, \ldots, A'_{m_n}], T'_p\} \subseteq R'_n[Y'; B'_1, \ldots, B'_{m_n}], T'_p\}$, where (a) $X' \equiv (\text{resp. } Y'), A'_1, \ldots, A'_{m_n}$ and $B'_1, \ldots, B'_{m_n}$ corresponds to $X$ (resp. $Y$), $A_1, \ldots, A_{m_n}$ and $B_1, \ldots, B_{m_n}$, (b) $X'_p \equiv (\text{resp. } Y'_p)$ corresponds to those finite domain attributes in $R_n$ (resp. $R_n$), but not in $A_1, \ldots, A_{m_n}$ (resp. $B_1, \ldots, B_{m_n}$), and (c) $T'_p = \{t'_{p_1}, t'_{p_2}, t'_{p_3} \}$ such that for each attribute $A'$ in $A'_1, \ldots, A'_{m_n}$ or $B'_1, \ldots, B'_{m_n}$, (i) $t'_{p_1}[A'] = t_{p_1}[A]$ and $t'_{p_2}[A'] = t_{p_3}[A'] \equiv 'i'$ if $\text{dom}(A)$ is infinite, and (ii) $t'_{p_1}[A'] = \rho(t_{p_1}[A])$ and $t'_{p_2}[A'] = t'_{p_3}[A'] \equiv 'i'$ if $\text{dom}(A)$ is finite; and (iii) for the rest attributes $A'$ in $X'_p$ or $Y'_p$, $t'_{p_1}[A'] = t_{p_1}[A'] \equiv 'i'$, $t'_{p_2}[A'] = t_{p_2}[A'] \equiv 'i'$, and $t'_{p_3}[A'] = t_{p_3}[A'] \equiv 'i'$.

Finally, one can easily verify that $\Sigma \models \psi$ iff $\Sigma' \models \psi'$, i.e., $\Sigma' \vdash \psi'$ for each CIND $\psi' \in \Sigma'$. Following from this, the problem is EXPTIME-hard.

Implication analysis of CFD$^\rho$s and CIND$^\rho$s. When CFD$^\rho$s and CIND$^\rho$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 [3]. Since CFD$^\rho$s and CIND$^\rho$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$^\rho$s and CIND$^\rho$s is undecidable.

**Remarks.** Inference systems play an important role for the implication analyses [3]. For the inference system of CFD$^\rho$s and CIND$^\rho$s alone, we can readily extend the one for CFDs [22] and CINDs [30], respectively, by deliberately handling the entailment of ordered pattern values involved with built-in predicates and their interaction with the wildcard 'i'. The details are left to interested readers. Note that it is easy to know that the implication analysis of CFD$^\rho$s and CIND$^\rho$s together is not finitely axiomatizable by Corollary 9.

**Summary.** The complexity bounds for reasoning about CFD$^\rho$s and CIND$^\rho$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 [22], [30]. The results tell us the following.

1. Despite the increased expressive power, CFD$^\rho$s and CIND$^\rho$s do not complicate the static analyses in the general case: the satisfiability and implication problems for CFD$^\rho$s and CIND$^\rho$s have the same complexity bounds as their counterparts for CFDs and CINDs. That is, the increased expressive power of CFD$^\rho$s and CIND$^\rho$s comes at a price in this special case.

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

**5 Validation of CFD$^\rho$s and CIND$^\rho$s**

If CFD$^\rho$s and CIND$^\rho$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 relational schema for $i \in [1, n]$.

The error detection problem is stated as follows.

The **error detection problem** is to find, given a set of CFD$^\rho$s and CIND$^\rho$s defined on $\mathcal{R}$, and a database instance $D = (I_1, \ldots, I_n)$ of $\mathcal{R}$ as input, the subset $(I'_1, \ldots, I'_m)$ 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$^\rho$ or CIND$^\rho$ in $\Sigma$. We denote the set as $\text{vio}(D, \Sigma)$, referred to it as the **violation set of $D$ w.r.t. $\Sigma$**.

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

**Theorem 10.** Given a set $\Sigma$ of CFD$^\rho$s and CIND$^\rho$s defined on $\mathcal{R} = (R_1, \ldots, R_n)$ and a database instance $D$ of $\mathcal{R}$, a set of SQL queries can be automatically generated such that (a) the collection of the answers to the SQL queries in $D$ is $\text{vio}(D, \Sigma)$, and (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 $\Sigma$.

Let $\Sigma_{\text{cfd}}$ be the set of all CFD$^\rho$s in $\Sigma$ defined on the same relational schema $R_i$, and $\Sigma_{\text{cind}}$ the set of all CIND$^\rho$s in $\Sigma$ from $R_i$ to $R_j$, for $i, j \in [1, n]$. We show the following. (a) The violation set $\text{vio}(D, \Sigma_{\text{cfd}})$ can be computed by two SQL queries. (b) Similarly, $\text{vio}(D, \Sigma_{\text{cind}})$ can be computed by a
These SQL queries use pattern tableaux of CFDP's (CINDP's) encoded with data tables, and hence their sizes are independent of $\Sigma$. From these Theorem 10 follows immediately.

We next present the main techniques for the query generation method, and the key idea is to encode CFDP's and CINDP's with data tables so that data dependencies and data themselves are uniformly represented, and SQL queries are then automatically generated to detect those tuples that violate certain CFDP's or CINDP's.

5.1 Encoding CFDP's and CINDP's with Data Tables

We first show the following, by extending the encoding of [10], [22]. The pattern tableaux of all CFDP's in $\Sigma_{\text{CFDP}}$ can be encoded with three data tables, and the pattern tableaux of all CINDP's in $\Sigma_{\text{CINDP}}$ can be represented as four data tables, no matter how many dependencies are in the sets.

Encoding CFDP's. We encode all pattern tableaux in $\Sigma_{\text{CFDP}}$ with three tables encL, encR, and encφ, where encL (resp. encR) encodes the non-negation $(\neq, \leq, \geq)$ patterns in LH5 (resp. RH5), and encφ encodes those negation $(\neq)$ patterns. More specifically, we associate a unique id $i$ with each CFDP's in $\Sigma_{\text{CFDP}}$, and let encL consist of the following attributes: (a) $\psi$, (b) each attribute $A$ appearing in the LH5 of some CFDP's in $\Sigma_{\text{CFDP}}$, and (c) its four companion attributes $A_{\leq}$, $A_{\geq}$, $A_{\neq}$, and $A'_{\neq}$. That is, for each attribute, there are five columns in encL, one for each non-negation operator. Similarly, encR is defined. We use an encφ tuple to encode a pattern $A \neq c$ in a CFDP, consisting of $\psi$, att, pos, and val, encoding the CFDP id, the attribute $A$, the position ('LHS' or 'RHS'), and the constant $c$, respectively. Note that the arity of encL (encR) is bounded by $5 \star |R_i| + 1$, where $|R_i|$ is the arity of $R_i$, and the arity of encφ is 4.

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

We populate encL, encR and encφ as follows. For each CFDP $\varphi = R(X \rightarrow Y; T_p)$ in $\Sigma_{\text{CFDP}}$, we generate a distinct id $i$ for it, and do the following.

(1) Add a tuple $t_1$ to encL such that (a) $t_1[\psi] = id_i$; (b) for each $A \in X$, $t_1[A] = \bot$, and for each non-negation predicate op, $t_1[\alpha_{\text{op}}] = \top$ if $T_p^{(\text{op}, A)} = \bot$; (c) we let $t_1[B] = \text{null}$ for all other attributes $B$ in encL.

(2) Similarly add a tuple $t_2$ to encR for attributes in $Y$.

(3) For each attribute $A \in X \cup Y$ and each $\neq$ pattern in $T_p[A]$, add a tuple $t$ to encφ such that $t[\psi] = id_i$, $t[\text{att}] = A$, $t[\text{val}] = \alpha$, and $t[\text{pos}] = \text{LHS}$ (resp. $t[\text{pos}] = \text{RHS}$) if attribute $A$ appears in $X$ (resp. $Y$).

Example 8: Recall from Fig. 2 CFDP's $\varphi_2$, $\varphi_3$, and $\varphi_4$ defined on relation item. The three CFDP's are encoded with the tables shown in Fig. 4: (a) $\text{encL}$ consists of attributes: $\text{cid}, \text{sale}, \text{price}, \text{price}_2$, and $\text{price}_2$; (b) $\text{encR}$ consists of $\text{cid}$, $\text{shipping}$, $\text{price}, \text{price}_2$, and $\text{price}_2$; those attributes in a table with only 'null' pattern values do not contribute to error detection, and are thus omitted; And (c) $\text{encφ}$ is empty since all these CFDP's have no negation patterns. One can easily reconstruct these CFDP's from tables encL, encR and encφ by collating the tuples based on $i$.

Encoding CINDP's. All CINDP's in $\Sigma_{\text{CINDP}}$ can be encoded with four tables encL, encL, encR, and encφ. Here encL (resp. encR) and encφ encode non-negation patterns on relation $R_i$ (resp. $R_j$) and negation patterns on relations $R_i$ and $R_j$, respectively, along the same lines as their counterparts for CFDP's. We use enc to encode the INDs embedded in CINDP's, which consists of the following attributes: (1) $\psi$ representing the id of a CINDP, and (2) those $X$ attributes of $R_i$ and $Y$ attributes of $R_j$ appearing in some CINDP's in $\Sigma_{\text{CINDP}}$. Note that the number of attributes in enc is bounded by $|R_i| + |R_j| + 1$, where $|R_i|$ is the arity of $R_i$.

For each CINDP $\psi = R_i[A_1 \ldots A_m; X_P; Y_P]$, $T_p$, $T_p$, in $\Sigma_{\text{CINDP}}$, we generate a distinct cid $id_i$ for it, and do the following.

(1) Add tuples $t_1$ and $t_2$ to encL and encR based on attributes $X_P$ and $Y_P$, respectively, along the same lines as their CFDP counterpart.

(2) Add tuples to encφ in the same way as their CFDP counterparts.

(3) Add tuple $t$ to enc such that $t[\psi] = id_i$. For each $k \in [1, n]$, let $t[A_k] = t[B_k] = k$, and $t[A] = \text{null}$ for the rest attributes $A$ of enc.

Example 9: Figure 5 shows the coding of CINDP's $\psi_1$ and $\psi_2$ given in Fig. 3. We use stateL and stateR in enc to denote the occurrences of attribute state in item and tax, respectively. In encL and encR, the attributes with only 'null' patterns are omitted, for the same reason as CFDP'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 $\Sigma$, regardless of the size of $\Sigma$. Recall that $n$ is the number of relations in the 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 $\text{vio}(D, \Sigma_{\text{CFDP}})$. 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 $\text{vio}(D, \Sigma_{\text{CINDP}})$. Putting these query answers together, we get $\text{vio}(D, \Sigma)$, the violation set of $D$ w.r.t. $\Sigma$.

SQL queries for CINDP's. Below we show how the SQL query $Q_{i,j}$ is generated for validating CINDP's in $\Sigma_{\text{CINDP}}$, which has not been studied by previous work. For the lack of space, we put the generation of detection queries for CFDP's in the supplementary material, which is an extension of the SQL techniques for CFD's and eCFD's discussed in [22] and [10], respectively.
The query $Q_{(i,j)}$ for the validation of $\Sigma_{cind}^{(i,j)}$ is given as follows, which capitalizes on the data tables $\text{enc}$, $\text{enc}_L$, $\text{enc}_R$ and $\text{enc}_{\rho}$ that encode CINDs in $\Sigma_{cind}^{(i,j)}$.

1. $R_i \times X \lor X \times N$ is the conjunction of $L.A_k = \text{null}$ or $R_i.A_k = L.A_k$ or $(L.A_k = '1' \lor L.A_k = '1')$ and $(L.A_k = '1' \lor L.A_k = '1')$ and $(L.A_k = '1' \lor L.A_k = '1')$ and $(L.A_k = '1' \lor L.A_k = '1')$ for each $k \in [1,m_1]$.

2. $R_j \times Y \lor Y \times N$ is defined similarly for attributes in $Y$.

3. $R_i \times X \lor X \times N$ is a shorthand for the conjunction below, for each $k \in [1,m_1]$:

   $(R_i.X = R_j.Y \land L.A_k = \text{null} \lor R_i.A_k = L.A_k \lor (L.A_k = '1' \lor L.A_k = '1'))$ and $(L.A_k = '1' \lor L.A_k = '1')$ and $(L.A_k = '1' \lor L.A_k = '1')$ and $(L.A_k = '1' \lor L.A_k = '1')$ for each $k \in [1,m_1]$.

4. $R_i \times X \lor X \times N$ is defined similarly, but with $N.pos = 'RHS'$; $H.A_k$ is null or $H.B_i$ is null or $H.A_k \neq H.B_i$.

5. $R_j \times Y \lor Y \times N$ is defined similarly for attributes in $Y$.

6. $R_i \times X \lor X \times N$ is a shorthand for the conjunction below, for each $k \in [1,m_1]$:

   $(R_i.X = R_j.Y \land L.A_k = \text{null} \lor R_i.A_k = L.A_k \lor (L.A_k = '1' \lor L.A_k = '1'))$ and $(L.A_k = '1' \lor L.A_k = '1')$ and $(L.A_k = '1' \lor L.A_k = '1')$ and $(L.A_k = '1' \lor L.A_k = '1')$ for each $k \in [1,m_1]$.

Example 10: Using the coding of Fig. 5, an SQL query $Q$ for checking CINDs $\psi_1$ and $\psi_2$ of Fig. 3 is given as follows:

```
select $R_i \times *$ from item $R_i$, $\text{enc}_L$, $\text{enc}_R$, $\text{enc}_{\rho}$
where $(L.type = \text{null} \lor L.type = 'L') \land \text{not exist (select * from N)}$
   where $N.cid = L.cid \land N.pos = 'LHS'$ and
   $(L.state = \text{null} \lor L.state = 'L') \land \text{not exist (select * from N)}$
   where $N.cid = L.cid \land N.pos = 'LHS'$ and
   $(L.att = \text{null} \lor L.att = 'L') \land \text{not exist (select * from N)}$
   where $N.cid = L.cid \land N.pos = 'LHS'$ and
   $(L.type = \text{null} \lor L.type = 'L') \land \text{not exist (select * from N)}$
   where $N.cid = L.cid \land N.pos = 'LHS'$ and
   $(L.state = \text{null} \lor L.state = 'L') \land \text{not exist (select * from N)}$
   where $N.cid = L.cid \land N.pos = 'LHS'$ and
   $(L.att = \text{null} \lor L.att = 'L')$
```

The SQL queries generated can be simplified as follows. As shown in Example 10, when checking patterns imposed by $\text{enc}_L$, $\text{enc}_R$, 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 $\text{enc}_{\rho}$, 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 $\text{enc}$, $\text{enc}_L$, $\text{enc}_R$ when encoding CINDs or CFPDs.

6 EXPERIMENTAL STUDY

We next present an extensive experimental study of CFPDs and CINDs. Using real-life data, we conducted two sets of experiments to evaluate the efficiency and effectiveness of CINDs and CFPDs vs. their counterparts CFDs and CINDs, separately and taken together.

6.1 Experimental Settings

We first present our experimental settings.

Datasets. We used two real-life datasets that were stored in an SQL Server 2012 database.

(1) Hospital Compare is a database publicly available from U.S. Department of Health & Human Services [1]. We used two tables hcahps and hcahps-state, which record the hospital level and state level ratings of the Hospital Consumer Assessment of Healthcare Providers and Systems (HCAHPS), respectively. For table hcahps, it records (a) the hospital information: hid (hospital ID), hname (hospital name), addr (address), city, state, zip, county, phn (phone number), and (b) the measure information: m (measure ID), m (question), mad (answer description), map (answer percentage), mnc (number of completed surveys), m (survey response rate percentage), mfn (footnote). And for table hcahps-state, it records state level measure information: state, m, m and map, among other things.

We designed 6 CFPDs and 3 CINDs for Hospital Compare, shown below in an informal way for easy of understanding:

```
\psi_1: hcahps (zip = '1' and city = '1' \lor county = '1' \land addr = '1')
\psi_2: hcahps (hid = '1' \lor hname = '1' \land count = '1' \land phn = '1')
\psi_3: hcahps (hid = '1' \lor msrp = '1')
```

7 CONCLUSION

In this paper, we have presented a novel approach to extend CFPDs and CINDs by encoding CFPDs and CINDs in the queries.
For comparison, we also designed the CFDs and CINDs counterparts of the above CFD5s and CIND5s. Here $\phi_1$, $\phi_2$, and $\psi_3$ are indeed CFDs and CINDs, respectively, while $\phi_3$, $\phi_4$, and $\psi_5$ are not. We hence further designed $\phi_5$, $\phi_6$, and $\psi_4$ to approximate $\phi_3$, $\phi_4$, and $\psi_5$, respectively.

(2) DBLP is a repository of computer science publications from 1946 to 2014 [2]. We further transformed its XML format into two tables paper and proceeding that record the paper and proceeding information, respectively, such that paper (key, year, title, booktitle, isbn, publisher) records books, journal articles and conference papers, and proceeding (key, year, isbn, publisher) records the proceedings of conference papers, respectively.

We generated 3482 CFD5s and 2568 CIND5s for the DBLP data, with their representatives shown below.

```
$\phi_1$: paper (isbn = 'l1' and booktitle = 'l2' and year = 'l3' and publisher = 'l4')
$\phi_2$: paper (title = 'l5' and booktitle = 'l6' and year = 'l7' and type = 'l8')
$\phi_3$: paper (booktitle = 'l9DB' and year = 'l10')
$\phi_4$: paper (booktitle = 'l11DB' and year = 'l12')
$\phi_5$: paper (booktitle = 'l13DB' and year = 'l14')
$\phi_6$: paper (booktitle = 'l15DB' and year = 'l16')
$\psi_1$: paper (crossref, isbn, publisher, type = 'l17' and booktitle = 'l18')
$\psi_2$: paper (crossref, isbn, publisher, type = 'l19' and booktitle = 'l20')
$\psi_3$: paper (crossref, isbn, publisher, type = 'l21' and booktitle = 'l22')
$\psi_4$: paper (crossref, isbn, publisher, type = 'l23' and booktitle = 'l24')
$\psi_5$: paper (crossref, isbn, publisher, type = 'l25' and booktitle = 'l26')
```

We collected all the booktitle and corresponding year from DBLP to generate the other CFD5s and CIND5s by instantiating the values of their booktitle and year attributes. Observe that $\phi_1$-$\phi_3$ and $\rho_1$ are CFDs and CINDs, respectively. For comparison, we further designed the following CFDS and CINDs to approximate $\phi_4$-$\phi_5$ and $\rho_2$-$\rho_3$.

```
$\phi_5$: paper (booktitle = 'l27DB' and year = 'l28' and publisher = 'l29')
$\phi_6$: paper (booktitle = 'l30DB' and year = 'l31' and publisher = 'l32')
$\rho_2$: paper (crossref, isbn, publisher, type = 'l33' and booktitle = 'l34')
$\rho_3$: paper (crossref, isbn, publisher, type = 'l35' and booktitle = 'l36')
```

Implementation. All the experiments were run within an SQL Server 2012 database installed on a machine with an Intel Core i5 (3.1GHz) CPU and 8GB of RAM. Each test was repeated 5 times, and the average is reported here.

6.2 Experimental Results

We next present our findings. Three parameters were used in our tests: (1) $|I_1|$, the number of tuples in table hcahs of HOSP or paper of DBLP, (2) $|I_2|$, the number of tuples in table hcahs-state of HOSP or proceeding of DBLP, and (3) noise%, the percentage of dirty tuples in table hcahs of HOSP or paper of DBLP, ranging from 0% to 9%. For easy of comparison, we deliberately dirty the tuples in hcahs of HOSP or paper of DBLP so that using the CFD5s and CIND5s together can detect all the dirty tuples. A clean copy of HOSP and DBLP is also kept to tell whether a tuple is dirty or clean.

6.2.1 Tests of Efficiency

In the first set of experiments, we evaluated the violation detection efficiency of CFD5s and CIND5s vs. their counterparts CFDS and CINDS, separately and taken together.

Exp-1.1: CFD5s vs. CFDS. (1) To evaluate the impacts of $|I_1|$, we fixed noise% = 9%, and varied $|I_1|$ from 10K to 90K for HOSP (resp. from 100K to 900K for DBLP); And (2) to evaluate the impacts of noise%, we fixed $|I_1| = 90K$ for HOSP (resp. 900K for DBLP), and varied noise% from 0% to 9%. The results are reported in Figures 6(a) and 6(c) and Figures 6(b) and 6(d), respectively.

The results tell us that for CFDS and CFD5s, both their running time (a) increases with the increment of the size of $I_1$, and (b) is insensitive to the noise. Furthermore, (c) their running time is mainly affected by three factors: the size of $I_1$, the LHS and RHS complexity of dependencies. For instance, (a) the LHS complexity of CFDS $\psi_3$ and $\psi_6$ is higher than CFD5s $\psi_5$ and $\psi_6$, as they match more $I_1$ tuples, but the RHS complexity of CFDS $\psi_3$ and $\psi_6$ is lower than CFD5s $\psi_5$ and $\psi_6$, as they are easier to check violations; And (b) the LHS complexity of CFDS $\phi_4$ and $\phi_6$ is the same as CFD5s $\phi_4$ and $\phi_6$, but the RHS complexity of CFDS $\phi_4$ and $\phi_6$ is similar to CFD5s $\phi_5$ and $\phi_5$, as they are easier to check violations. As a combined result, the running time of CFDS is lower than CFD5s on HOSP, but close to CFD5s on DBLP.

Exp-1.2: CIND5s vs. CINDS. (1) To evaluate the impacts of $|I_1|$, we fixed noise% = 9% and $|I_2| = 1.6K$ for HOSP (resp. 16K for DBLP), and varied $|I_1|$ from 10K to 90K for HOSP (resp. from 100K to 900K for DBLP); (2) To evaluate the impacts of $|I_2|$, we fixed noise% = 9% and $|I_1| = 90K$ for HOSP (resp. 900K for DBLP), and varied $|I_2|$ from 1K to 1.6K for HOSP (resp. from 10K to 16K for DBLP); And (3) To evaluate the impacts of noise%, we fixed $|I_1| = 90K$ for HOSP (resp. 900K for DBLP) and $|I_2| = 1.6K$ for HOSP (resp. 16K for DBLP), and varied noise% from 0% to 9%. The results are reported in Figures 7(a) and 7(d), Figures 7(b) and 7(e), and Figures 7(c) and 7(f), respectively.

The results tell us that for CINDs and CIND5s, both their running time (a) increases with the increment of the size of $I_1$, (b) is not affected much by $I_2$ as $|I_2|$ is relatively small in the tests, and (c) is insensitive to the noise. Furthermore, (d) their running time is mainly affected by four factors: the size of $I_1$, the size of $I_2$, the LHS and RHS complexity of dependencies. For instance, (a) the LHS complexity of CIND $\psi_3$ is higher than CIND5 $\psi_3$, as they match more $I_1$ tuples, but the RHS complexity of CIND $\psi_3$ is lower than CIND5 $\psi_3$, as they are easier to check violations; And (b) the LHS complexity of CINDS $\rho_2$ and $\rho_3$ is the same as CIND5s $\rho_2$ and $\rho_3$, but the RHS complexity of CINDS $\rho_2$ and $\rho_3$ is lower than CIND5s $\rho_2$ and $\rho_3$, as they are easier to check violations. As a combined result, the running time of CINDs is close to CIND5s on HOSP, but is lower on DBLP.
The results show similar findings to Exp-1.1 and Exp-1.2, Figures 8(b) and 8(e) and Figures 8(c) and 8(f), respectively.

Figure 6. Efficiency of detecting violations: CFD’s vs. CFDs

Figure 7. Efficiency of detecting violations: CIND’s vs. CINDs

Figure 8. Efficiency of detecting violations: CFD’s + CIND’s vs. CFDs + CINDs

noise%. The results are reported in Figures 8(a) and 8(d), Figures 8(b) and 8(e) and Figures 8(c) and 8(f), respectively. The results show similar findings to Exp-1.1 and Exp-1.2, and are consistent with them.

6.2.2 Tests of Effectiveness

In the second set of experiments, we evaluated the violation detection effectiveness of CFD’s and CIND’s vs. their counterparts CFDs and CINDs, separately and taken together. Note that we did not report the results of varying $|I_2|$ as it has no impacts on the effectiveness tests in our setting.

Given one of CFDs, CFD’s, CINDs, CIND’s, CFDs + CINDs or CFD’s + CIND’s, denoted by $x$, its effectiveness of detecting violations is evaluated with the following measure:

$$\text{accuracy}(x) = \frac{\#\text{dirty tuples found by } x}{\#\text{dirty tuples found by cfd’s + cind’s}}.$$  

Exp-2. Using the same setting as Exp-1.1, Exp-1.2 and Exp-1.3, respectively, we evaluated the impacts of $|I_1|$ and noise% for (a) CFD’s vs. CFDs, (b) CIND’s vs. CINDs and (c) CFD’s + CIND’s vs. CFDs + CINDs, respectively. The results are reported in Figures 9, 10 and 11, respectively, and are summarized in Table 2.

The results tell us that (1) the effectiveness of detecting violations using all classes of dependencies are robust to $|I_1|$ and noise%, (2) CFD’s, CIND’s and CFD’s + CIND’s obviously outperform their counterparts CFDs, CINDs and CFDs + CINDs, respectively, (3) the increase of effectiveness
depends on the increase of the expressive power, and varies from 22% to 75% on HOSP and DBLP, and, (4) the increased effectiveness on DBLP is larger than on HOSP, as there are more CFDp’s and CINDp’s on HOSP that can be expressed by CFDs and CINDs than on DBLP in our tests.

Summary. From these experimental results on real-life data HOSP and DBLP, we find the following. (1) The running time of CFDp’s and CINDp’s is comparable to their CFDs and CINDs counterparts, which is consistent with the static analyses: CFDp’s and CINDp’s retain the same complexity as their CFDs and CINDs counterparts. (2) CFDp’s and CINDp’s are able to capture more dirty tuples than CFDs and CINDs, due to the increased expressive power.

7 Conclusions
We have proposed CFDp’s and CINDp’s, which further extend CFDs and CINDs, respectively, by allowing patterns on data values to be expressed in terms of ≠, <, ≤, > and ≥ predicates. We have shown that CFDp’s and CINDp’s are more powerful than CFDs and CINDs for detecting errors in real-life data. In addition, the satisfiability and implication problems for CFDp’s and CINDp’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 CFDp’s and CINDp’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., CFDp’s, eCFDs [10] and CFDp’s [13]), without increasing the complexity of static analyses. Second, we are to develop effective algorithms for discovering CFDp’s and CINDp’s, along the same lines as [5], [28], [36]. Third, we plan to extend the methods of [8], [17] to repair data based on CFDp’s and CINDp’s, instead of using CFDs [17], traditional FDs and INdS [8], denial constraints [7], and aggregate constraints [25].

Acknowledgments
Ma is supported in part by 973 program 2014CB340300 and NSFC 61322207. Fan is supported in part by NSFC.
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKE.2015.2451632, IEEE Transactions on Knowledge and Data Engineering

61133002, 973 Program 2014CB340302, Shenzhen Peacock Program 1105100030834361, Guangdong Innovative Research Team Program 2011D005, EPSRC EP/J015377/1 and EP/M025268/1, and a Google Faculty Research Award.

**References**


**Shuai Ma** is a professor at the School of Computer Science and Engineering, Beihang University. He obtained his PhD degrees from University of Edinburgh in 2010, and from Peking University in 2004, respectively. He was a post-doctoral research fellow in the database group, University of Edinburgh, a summer intern at Bell labs, Murray Hill, USA, in the summer of 2008, and a visiting researcher of MSR Institute in 2012. He is a recipient of the Best Paper Award for VLDB 2010 and the Best Challenge Paper Award for ICDE 2007. His current research interests include database theory and systems, social data analysis and data intensive computing.

**Liang Duan** is a PhD student in the School of Computer Science and Engineering, Beihang University, co-supervised by Prof. Jinpeng Huai and Prof. Shuai Ma. He received his MS degree in computer software and theory from Yunnan University in 2014, and BS degree in computer science and technology from Beihang University in 2009. His current research interests include databases and social data analysis.

**Wenfei Fan** is Professor (Chair) of Web Data Management in the School of Informatics, University of Edinburgh, UK. He is a Fellow of ACM, a Fellow of the Royal Society of Edinburgh, UK, a National Professor of the Thousand-Talent Program and a Yangtze River Scholar, China. He received his PhD degree from the University of Pennsylvania. He is a recipient of the Runner-up for Best Paper Award of ICDE 2014, the Alberto O. Mendelzon Test-of-Time Award of ACM PODS 2010, the Best Paper Award for VLDB 2010, the Roger Needham Award in 2008 (UK), the Best Paper Award for ICDE 2007, the Best Paper of the Year Award for Computer Networks in 2002, and the Career Award in 2001 (USA). His current research interests include database theory and systems.

**Chunning Hu** is an associate professor at the School of Computer Science and Engineering, Beihang University. He received his PhD degree from Peking University in 2006. His current research interests include distributed systems, system virtualization, large scale data management and processing systems.

**Wenguang Chen** is an associate professor at the Department of Computer Science, Peking University. He obtained his PhD degree from Peking University in 2009. He was a visiting scholar at University of Alberta from 2011 to 2012 and at University of Hawaii at Manoa in from 2012 to 2013. His current research interests include data management, data quality and intelligent HCI.