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 CFDps and CINDps, respectively, by specifying patterns of data values with \( \neq, <, \leq, > \), and \( \geq \) predicates. As data quality rules, CFDps and CINDps 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 CFDps and CINDps. (a) One concerns the satisfiability and implication problems for CFDps and CINDps, 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 CFDps and CINDps retain the same complexity as their CFDs and CINDs counterparts. (b) The other concerns validation of CFDps and CINDps. We show that given a set \( \Sigma \) of CFDps and CINDps 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 CFDps and CINDps.

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:

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

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:

\( \text{cfd}_1: \text{item} (\text{id} \rightarrow \text{name}, \text{type}, \text{price}, \text{shipping}, \text{sale}) \)
\( \text{cfd}_2: \text{tax} (\text{state} \rightarrow \text{rate}) \)
\( \text{cfd}_3: \text{item} (\text{sale} = 'T' \rightarrow \text{shipping} = 0) \)

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 \( t_i \), if \( t_i[\text{sale}] = 'T' \) then \( t_i[\text{shipping}] \) 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, sale = ‘T’ and shipping = 0. It is to hold only on item tuples that match the pattern 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:

\( \text{cfd}_4: \text{item} (\text{id} \rightarrow \text{name}, \text{type}, \text{price}) \)
\( \text{cfd}_5: \text{tax} (\text{state} \rightarrow \text{rate}) \)
\( \text{cfd}_6: \text{item} (\text{sale} = 'T' \rightarrow \text{shipping} = 0) \)

S. Ma, L. Duan and C. Hu are with the SKLSDE lab, School of Computer Science and Engineering, Beihang University, China. E-mail: {mashuai, duanl, luchm}@act.buaa.edu.cn.

W. Fan is with the RCRD center, Beihang University, China and the School of Informatics, Edinburgh University, UK. E-mail: wenfei@inf.ed.ac.uk.

W. Chen is with the Department of Information Management, Peking University, China. E-mail: chenwq@pku.edu.cn.

Manuscript received XXX, 2014; revised XXX, 2014.
The 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/TKDE.2015.2451632, IEEE Transactions on Knowledge and Data Engineering

Figure 1. Example instance \( D_0 \) of item and tax

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>type</th>
<th>price</th>
<th>shipping</th>
<th>sale</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>b1</td>
<td>book</td>
<td>25.99</td>
<td>0</td>
<td>T</td>
<td>WA</td>
</tr>
<tr>
<td>t2</td>
<td>c1</td>
<td>CD</td>
<td>9.99</td>
<td>2</td>
<td>F</td>
<td>NY</td>
</tr>
<tr>
<td>t3</td>
<td>b2</td>
<td>book</td>
<td>34.99</td>
<td>20</td>
<td>F</td>
<td>DL</td>
</tr>
<tr>
<td>t4</td>
<td>a1</td>
<td>art</td>
<td>5m</td>
<td>500</td>
<td>F</td>
<td>DL</td>
</tr>
</tbody>
</table>

(a) An item relation

<table>
<thead>
<tr>
<th>state</th>
<th>rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>t5</td>
<td>PA</td>
</tr>
<tr>
<td>t6</td>
<td>NY</td>
</tr>
<tr>
<td>t7</td>
<td>NY</td>
</tr>
<tr>
<td>t8</td>
<td>NJ</td>
</tr>
</tbody>
</table>

(b) A tax rate relation

pf\(_d_1\): item (sale = 'F' and price \( \leq 20 \) → shipping = 3)

pf\(_d_2\): item (sale = 'F' and price > 20 and price \( \leq 40 \) → shipping = 6)

pf\(_d_3\): item (sale = 'F' and price > 40 → shipping = 10)

pf\(_d_4\): item (sale = 'T' → price \( \geq 2.99 \) and price < 9.99)

p\(_\text{ind}_1\): item (state; type = 'art') ≤ tax (state; nil)

Here pf\(_d_2\) states that for any item tuple, if it is not on sale and its price is in the range \([20, 40]\), then its shipping fee must be 6; similarly for pf\(_d_1\) and pf\(_d_3\). These dependencies extend CFD\(_s\) [22] by specifying patterns of semantically related data values in terms of predicates \( <, \leq, > \) and \( \geq \). Similarly, pf\(_d_4\) assures that for any item tuple, if it is on sale, then its price must be in the range \([2.99, 9.99]\). Finally, p\(_\text{ind}_1\) extends CIND\(_s\) [30] by specifying patterns with \( \neq \): for any item tuple \( t \), if \( t(\text{type}) \) is not artwork, then there must exist a tax tuple \( t' \) such that \( t(\text{state}) = t'(\text{state}) \), i.e., the sale tax of the item can be found from the tax relation.

Using dependencies pf\(_d_1\)–pf\(_d_4\) and p\(_\text{ind}_1\) as data quality rules, we find that \( D_0 \) of Fig. 1 is not clean. Indeed, (a) \( t_2 \) violates pf\(_d_1\); its price is less than 20, but its shipping fee is 2 rather than 3; similarly, \( t_3 \) violates pf\(_d_2\) and \( t_4 \) violates pf\(_d_4\). (b) Tuple \( t_1 \) violates pf\(_d_4\); it is on sale but its price is not in the range \([2.99, 9.99]\). (c) The database \( D_0 \) also violates p\(_\text{ind}_1\); \( t_1 \) 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 pf\(_d_1\)–pf\(_d_4\) and p\(_\text{ind}_1\) can be expressed as FDS or INDS [3], which do not allows constants, or as CFD\(_s\) or CIND\(_s\) [30], which specify patterns with equality (=) only. While there have been extensions of CFD\(_s\) [10], [13], [28], none of these allows dependencies to be specified with patterns on data values in terms of built-in predicates \( \neq, <, \leq, >, \geq \). To the best of our knowledge, the earlier conference version [12] of this paper is the first to study these constraints. These highlight the need for extending CFD\(_s\) and CIND\(_s\) to capture errors commonly found in real-life data. While one can consider arbitrary extensions, it is necessary to strike a balance between their expressive power and their complexity. In particular, we want to be able to reason about data quality rules expressed as extended CFD\(_s\) and CIND\(_s\). Furthermore, we want to have effective algorithms to detect inconsistencies based on these extensions.

**Contributions & Roadmap.** To this end we introduce an extension of CFD\(_s\) and CIND\(_s\), investigate the static analyses of these constraints, and develop effective SQL-based techniques for detecting errors based on these constraints.

(1) We propose two classes of dependencies, denoted by CFD\(_s\) and CIND\(_s\), which respectively extend CFD\(_s\) and CIND\(_s\) by supporting \( \neq, <, \leq, >, \geq \) predicates (Sections 2 and 3). For example, all the dependencies we have encountered so far can be expressed as CFD\(_s\) or CIND\(_s\). These dependencies are capable of capturing errors in real-world data that cannot be detected by CFD\(_s\) or CIND\(_s\).

(2) We establish the complexity bounds for the satisfiability and implication problems for CFD\(_s\) and CIND\(_s\), 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, CFD\(_s\) and CIND\(_s\) 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\(_s\), respectively, (b) in \( O(1) \)-time (constant-time) and EXPTIME-complete for CIND\(_s\), respectively, and (c) are undecidable when CFD\(_s\) and CIND\(_s\) 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 CFD\(_s\) and CIND\(_s\) (Section 5). Given a set \( \Sigma \) of CFD\(_s\) and CIND\(_s\) 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 \( \Sigma \). Further, the SQL queries are independent of the size and cardinality of \( \Sigma \). These provide the capability of detecting errors in a single relation (CFD\(_s\)) and across different relations (CIND\(_s\)) 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 CFD\(_s\) and CIND\(_s\) is comparable to their CFDs and CINDs counterparts, which is consistent with the static analyses in Section 4, and (b) CFD\(_s\) and CIND\(_s\) 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 CFD\(_s\) and CIND\(_s\), separately and taken together (Section 4), and (b) an extensive experimental study of CFD\(_s\) and CIND\(_s\) (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 coNP-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 CFPs 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 CFPs 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 ($\varphi \neq \bot, \preceq, \succ, \triangleright$, $\leq, \geq$), from static analyses to error detection.

Methods have been developed for discovering CFDs [14], [28], CFPs [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 CFPs 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 CFPs, constraint-generating dependencies (CGDs) of [6] and constrained tuple-generating dependencies (CTGDs) of [32] can express CFPs, 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 $=, \preceq, \succ, \triangleright$, matching dependencies [20] and differential dependencies [34] that support the comparison of attributes with $=, \neq, \preceq, \succ, \triangleright$ for record matching. However, different from CFDs and CFPs, 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].

\section{Extending CFDs with Predicates}

We now define conditional functional dependencies with predicates, denoted by CFPs, by extending CFDs [22] with built-in predicates ($\varphi \neq \bot, \preceq, \succ, \triangleright$) in addition to equality ($\equiv$).

Consider a relational schema $R$ defined over a finite set of attributes, denoted by $\text{attr}(R)$. 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., bool) or infinite (e.g., string). We assume w.l.o.g. that a domain on which $\preceq, \leq, >$ or $\geq$ is defined is totally ordered.

\textbf{Syntax}. A CFP $\varphi$ on $R$ is a pair $R(X \rightarrow Y, T_p)$, where (1) $X, Y$ are sets of attributes in $\text{attr}(R)$; (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 $t_A \in T_p, t_A[A]$ is either an unnamed variable ‘$\_\_$’ that draws values from $\text{dom}(A)$, or ‘op $a$’, where op is one of $=, \neq, \preceq, \succ, \triangleright$, 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 $R$ is clear from the context, and denote $X$ as $\text{LHS}(\varphi)$ and $Y$ as $\text{RHS}(\varphi)$, respectively.

\textbf{Example 2}: The dependencies $\text{cd}_1 \rightarrow \text{cd}_3$ and $\text{pd}_1 \rightarrow \text{pd}_4$ that we have seen in Example 1 can all be expressed as CFPs. One of these CFPs is illustrated in Fig. 2, in which $\varphi_1$ is for FD $\text{cd}_2$, $\varphi_2$ is for CFD $\text{cd}_3$, $\varphi_3$ is for $\text{pd}_2$, and $\varphi_4$ is for $\text{pd}_4$, respectively.

\textbf{Semantics}. Consider CFP $\varphi = R(X \rightarrow Y, T_p)$, where $T_p = \{t_1, \ldots, t_k\}$.

A data tuple $t$ of $R$ is said to match $\text{LHS}(\varphi)$, denoted by $t[X] \models T_p$, if for each tuple $t_i \in T_p, \varphi$ in $T_p$ and each
attribute A in X, either (a) $t_p[A] = \text{wildcard} \ 'c' \ (which \ matches \ any \ value \ in \ \text{dom}(A))$, or (b) $t[A] \ op \ a$ if $t_p[A] = \text{op} a$, where the operator op $= (\neq, \leq, \geq, >) \ op$ is interpreted by its standard semantics. Similarly, the notion that $t$ matches RHS($\varphi$) is defined, denoted by $t[Y] \triangleright \text{RHS}(\varphi)$.

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

An instance $I$ of $R$ satisfies the CFD$^p$ $\varphi$, denoted by $I \models \varphi$, if for each pair of tuples $t_1, t_2$ in $I$, if $t_1[X] = t_2[X] \triangleright \text{RHS}(\varphi)$, then $t_1[Y] = t_2[Y] \triangleright \text{RHS}(\varphi)$. 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 $\varphi$ is imposed only on the subset of tuples in $I$ that match 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) $t_1[Y] \triangleright \text{RHS}(\varphi)$, which assures that the constants in $t_1[Y]$ match the constants in $t_2[Y]$ for all $t_p$ 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 $I \models \varphi$ for each CFD$^p$ $\varphi$ in $\Sigma$.

Example 3: The instance $D_0$ of Fig. 1 satisfies $\varphi_1$ and $\varphi_2$ of Fig. 2, but neither $\varphi_3$ nor $\varphi_4$. Indeed, tuple $t_3$ violates (i.e., does not satisfy) $\varphi_3$, since $t_3[\text{sale}] = 'F'$ and $t_3[\text{price}] \leq 40$, but $t_3[\text{shipping}] = 20$ instead of 6. Note that $t_4$ matches LHS($\varphi_3$) since it satisfies the condition specified by the conjunction of the pattern tuples in $T_3$. Similarly, $t_1$ violates $\varphi_4$, since $t_1[\text{sale}] = 'T'$ 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 $(X \rightarrow Y, T_p)$ in which $T_p$ contains a single tuple consisting of ‘$\top$’ only, without constants. (2) A CFD $(X \rightarrow Y, T_p)$ [22] with $T_p = \{t_1, \ldots, t_p\}$ can be expressed as a set $\{\varphi_1, \ldots, \varphi_k\}$ of CFD$^p$s such that for each $i \in [1,k]$, $\varphi_i = (X \rightarrow Y, T_p)$, where $T_p$ contains the pattern tuple $t_p$ of $T_p$ only, defined with equality (=) only. For example, $\varphi_1$ and $\varphi_2$ in Fig. 2 are CFD$^p$s representing FD $c$fd$^p$ and CFD $c$fd$^3$ in Example 1, respectively. Note that all data quality rules in [14], [28] can be expressed as CFD$^p$s.

3 Extending CIND$^p$s 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, >, \geq) in addition to equality (=). Consider two relational schemas $R_1$ and $R_2$.

Syntax. A CIND$^p$ $\psi$ is a pair $(R_1[X: X_p] \subseteq R_2[Y: Y_p], T_p)$, where (1) $X, X_p$ and $Y, Y_p$ are lists of attributes in $\text{attr}(R_1)$ and $\text{attr}(R_2)$, 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 $X_p \cup Y_p$, and for each $A$ in $X_p$ or $Y_p$, each pattern tuple $t_p \in T_p$, $t_p[A]$ is either an unnamed variable ‘$\prime$’ that draws values from $\text{dom}(A)$, or ‘$\prime op a’ where op is one of $=, \neq, \leq, \geq, >$, and ‘$\prime$’ is a constant in $\text{dom}(A)$.

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 ‘$\prime$’. We also use nil to denote an empty list.

Example 4: Figure 3 shows two example CIND$^p$s: $\psi_1$ expresses the $\text{pind}_1$ in Example 1, and $\psi_2$ refines $\psi_1$ 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., $\psi_2$ 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$^p$ $\psi$, denoted by $(I_1, I_2) \models \psi$, if for each tuple $t_1 \in I_1$, if $t_1[X_p] \approx \text{RHS}(\psi)$, then there exists a tuple $t_2 \in I_2$ such that $t_1[X] = t_2[X]$ and $t_2[Y_p] \approx \text{RHS}(\psi)$.

That is, if $t_1[X]$ matches the pattern tableau $T_p[X]$, then $\psi$ assures the existence of $t_2$ such that (1) $t_1[X] = t_2[X]$ as needed by the standard IND embedded in $\psi$; and, moreover, (2) $t_2[Y_p]$ must match the pattern tableau $T_p[Y_p]$. 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 $T_p[Y_p]$ 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 all the pattern tuples in $T_p$.

Example 5: The instance $D_0$ of item tax in Fig. 1 violates CIND$^p$ $\psi_1$. Indeed, tuple $t_3$ 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$^p$ $(R_1[X: X_p] \subseteq R_2[Y: Y_p], T_p)$ 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 $C$ in the embedded IND and $t_p[C]$ and $t_p'[A]$ are not satisfiable when taken together; e.g., $t_p[\text{price}] = 9.99$ and $t_p'[\text{price}] \geq 19.99$.

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$^p$ $(R_1[X: nil] \subseteq R_2[Y: nil], T_p)$ such that $T_p$ is simply a empty set. (2) A CIND $(R_1[X: X_p] \subseteq R_2[Y: Y_p], T_p)$ with $T_p = \{t_1, \ldots, t_p\}$ can be expressed as a set $\{\psi_1, \ldots, \psi_k\}$ of CIND$^p$s, where for each $i \in [1,k]$, $\psi_i = (R_1[X: X] \subseteq R_2[Y: Y], T_p[i])$ such that $T_p[i]$ consists of the pattern tuple $t_p[i]$ of $T_p$, 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 $\Sigma$. 1041-4347 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
The satisfiability analysis of conditional dependencies is not only of theoretical interest, but is also important in practice. Indeed, when CFDs and CINDPSs 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 CFDs. 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 CFDs, the same problem exists when it comes to CFDs.

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 CFDs despite their increased expressive power.

**Proposition 1:** The satisfiability problem for CFDs is NP-complete.

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

We next present an NP algorithm that, given a set $\Sigma$ of CFDs 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 $R$ 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 CFDs 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 $b_{01} \in \text{dom}(A_i)$ to $C_1$ such that $b_{01} < a_1$ if there exists one; And also add another constant $b_{02} \in \text{dom}(A_i)$ to $C_1$ such that $b_{02} < a_1$ and $b_{02} \neq b_{01}$ if there exists one;
3. Similarly, for each $j \in [1, k - 1]$, add a constant $b_{j1} \in \text{dom}(A_i)$ to $C_1$ such that $a_j < b_{j1} < a_{j+1}$ if there exists one; And also add another constant $b_{j2} \in \text{dom}(A_i)$ to $C_1$ such that $a_j < b_{j2} < a_{j+1}$ and $b_{j2} \neq b_{j1}$ if there exists one;
4. Finally, add a constant $b_{k1} \in \text{dom}(A_i)$ to $C_1$ such that $b_{k1} < a_k$ if there exists one; And also add another constant $b_{k2} \in \text{dom}(A_i)$ to $C_1$ such that $b_{k2} > a_k$ and $b_{k2} \neq b_{k1}$ if there exists one.

Moveover, the number of elements in $\text{dom}(A_i)$ is at most $3 \ast h + 2$. Then one can easily verify that $\Sigma$ is satisfiable iff 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 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 CFDs. This tells us that the increased expressive power of CFDs 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 CFDs 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 $y_{i1} \lor y_{i2} \lor y_{i3}$ such that for each $i \in [1, 3]$, $y_{ij}$ is either $x_{pj}$ or $\overline{x_{pj}}$ for $p_j \in [1, m]$. Given $\phi$, we construct a relational schema $R$ and a set $\Sigma$ of CFDs defined on $R$ such that $\phi$ is satisfiable iff $\Sigma$ is satisfiable.

1. We first define the relational schema $R(X_1, \ldots, X_m, C_1, \ldots, C_n, Z)$, where all attributes share a common infinite domain $\mathbb{D}$ that contains constant $a$. Intuitively, for each $R$ tuple $t$, $\{x_1, \ldots, x_m\}$ specifies a truth assignment $\xi$ for variables $x_1, \ldots, x_m$ of $\phi$, and $t[C_i]$ ($i \in [1, n]$) and $t[Z]$ are the truth values of clause $C_i$ and sentence $\phi$ w.r.t. the assignment $\xi$, respectively.
2. We then construct the set $\Sigma$ of CFDs, which intuitively encode the relationships of the truth values between the clauses.
domains, we construct a decision procedure, by extending the proof of its counterpart in [30].

The implication analysis of CFDs is NP-complete.

Proof: The lower bound follows from the NP-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 $\phi = R(X \rightarrow Y, T_p)$ and $\Sigma \not\models \phi$, then there exists an instance $I$ of $R$ with two tuples $t_1$ and $t_2$ such that $I \models \Sigma$ and $t_1[X] = t_2[X] \Rightarrow T_p[X]$, but either $t_1[Y] \neq t_2[Y]$ or $t_1[Y] \neq T_p[Y]$ (resp. $t_2[Y] \neq T_p[Y]$). 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 $\Sigma \not\models \phi$ iff there exist two mappings $\rho_1$ and $\rho_2$ from all attributes $A_i$ to $adom(A_i)$ ($i \in [1, m]$) such that $I = \{\rho_1(A_1), \ldots, \rho_1(A_m), \rho_2(A_1), \ldots, \rho_2(A_m)\}, I \models \Sigma$, but $I \not\models \phi$.

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 $\phi = \{\text{item (sale, price } \rightarrow \text{ shipping,}$ $T\}$, where $T$ consists of a single pattern tuple (sale = 'F', price = 30 || shipping = 6). Thus in the presence of the CFDs in Fig. 2, $\phi$ is redundant.

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 its counterpart in [22].

Proposition 5: The implication problem for CFDs is conp-complete.
Based on these, we give an NP algorithm as follows: (1) Guess two \( R \) tuples \( t_1 \) and \( t_2 \) such that \( t_1[A]_i, t_2[A]_i \in \text{atom}(A_i) \) for each \( i \in [1, m] \). (2) Check whether \( I = \{ t_1, t_2 \} \) satisfies \( \Sigma \), but not \( \varphi \). 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 CINDs 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 CINDP, which does not find a counterpart in [22].

**Proposition 6:** In the absence of finite domain attributes, the implication problem for CINDP is \( \text{coNP} \)-complete.

**Proof:** The problem is in \( \text{coNP} \) by Proposition 5. The \( \text{coNP} \)-hardness is shown by reduction from the 3SAT problem to its complement problem, i.e., the problem for determining whether \( \Sigma \not\models \varphi \).

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

The relational schema \( R \) and the set \( \Sigma \cup \{ \varphi \} \) of CINDP are the same as the corresponding ones in Proposition 2. Moreover, \( \varphi \) is defined as \( (Z \to Z, T_p) \), where \( T_p = \{ (_{-} || \neq a) \} \). Intuitively, \( \varphi \) requires that for any \( R \) tuple \( t, t[Z] \neq a \). Along the same lines as Proposition 2, one can easily verify that \( \varphi \) is satisfiable iff \( \Sigma \not\models \varphi \). Thus the problem is \( \text{coNP} \)-hard.

**Implication analysis of CINDP**. We next show that CINDP do not make their implication analysis harder, verified by extending the proof of their CIND counterpart given in [30].

**Proposition 7:** The implication problem for CINDP is \( \text{EXPTIME} \)-complete.

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

We next present the \( \text{EXPTIME} \) algorithm. Consider \( \mathcal{R} = (R_1, \ldots, R_n) \) and \( \psi = (R_0[X_0; Y_0] \subseteq R_0[Y_0; T_p]) \). And for each attribute \( A \), define the active domain \( \text{atom}(A) \) 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 \( \text{EXPTIME} \) 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 \( t_i[A] \in \text{atom}(A) \) for each attribute \( A \in \text{attr}(R_i) \). There is an edge \( e = (\gamma_i = (R_i : t_i', R_j : t_j')) \) in \( E \) iff there exists a CINDP \( \phi = (R_0[U; U_p] \subseteq R_j[V; Y_j; T_p]) \) in \( \Sigma \) such that \( t_i[U] \equiv \gamma_i[U_p], t_j[V] = t_j[U] \) and \( t_j[V'] \equiv \gamma_i[V_p], \) and \( e \) is labeled with the CINDP \( \phi \), i.e., \( e \in \{(l)\} \). Note that an edge may have multiple labels.

2. Let \( S_\psi \) be the set of nodes ‘\( R_i : t_i' \)’ such that \( t_i[X] \equiv \gamma_i[Y] \), and \( S_\varphi \) be the set of nodes ‘\( R_i : t_i' \)’ such that \( t_i[Y] \equiv \gamma_i[X] \), respectively.

3. For each node \( u = \gamma_i : t_i' \) in \( S_\psi \), 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 \( u = \gamma_i : t_i' \), 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 node in ‘\( R_i : t_i' \)’ in \( S_\psi \) with \( t_i[X] = t_i[Y] \) 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 \( u = \gamma_i : t_i' \), we return ’no’, and return ’yes’, otherwise.

6. 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 whole 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 \( O(|\Sigma \cup \{ \psi \}| \cdot |\mathcal{R}|) \); (b) The number of nodes in sets \( S_\psi \) or \( S_\varphi \) is bounded by the maximum number of tuples in a database too; (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 \( \text{EXPTIME} \).

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 CINDP.

**Theorem 8:** In the absence of finite domain attributes, the implication problem for CINDP remains \( \text{EXPTIME} \)-complete.

**Proof:** The problem is in \( \text{EXPTIME} \) by Proposition 7. The \( \text{EXPTIME} \)-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 \( \text{EXPTIME} \)-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 \( \mathcal{R} \), we construct another database schema \( \mathcal{R}' \), in which each relation \( R'_i \) (\( i \in [1, n] \)) consists of infinite database attributes only, and a set \( \Sigma' \cup \{ \psi' \} \) of CINDP on \( \mathcal{R}' \) such that \( \Sigma \models \psi \) iff \( \Sigma' \models \psi' \).
Table 1

<table>
<thead>
<tr>
<th>Σ</th>
<th>General setting</th>
<th>Infinite domain only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Satisfiability</td>
<td>Implication</td>
</tr>
<tr>
<td>CFDS [22]</td>
<td>NP-complete</td>
<td>coNP-complete</td>
</tr>
<tr>
<td>CFDS’s</td>
<td>NP-complete</td>
<td>coNP-complete</td>
</tr>
<tr>
<td>CINDS [30]</td>
<td>O(1)</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>CINDS’s</td>
<td>O(1)</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>CFDS + CINDS [30]</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
<tr>
<td>CFDS + CINDS’s</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
</tbody>
</table>

5 Validation of CFDS’s and CINDS’s

If CFDS’s and CINDS’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 dependen-
cies, by leveraging functionality supported by commercial DBMSs. More specifically, consider a database schema R = (R1, . . . , Rn), where Ri is a relational schema for i ∈ [1, n]. The error detection problem is stated as follows.

The error detection problem is to find, given a set Σ of CFDS’s and CINDS’s defined on R, and a database instance D = (I1, . . . , In) of R as input, the subset (I′1, . . . , I′n) of D such that for each i ∈ [1, n], I′i ⊆ Ii and each tuple in I′i violates at least one CFDS or CINDS in Σ. We denote the set as vio(D, Σ), 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 CFDS’s and CINDS’s. The main result of the section is as follows.

Theorem 10: Given a set Σ of CFDS’s and CINDS’s defined on R = (R1, . . . , Rn) and a database instance D of 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 vio(D, Σ), and (b) the number and size of the set of SQL queries depend only on the number n of relations and their arities in R, regardless of Σ.

Let Σcfds be the set of all CFDS’s in Σ defined on the same relational schema R, and Σcind be the set of all CINDS’s in Σ from Ri to Rj for i, j ∈ [1, n]. We show the following. (a) The violation set vio(D, Σcfds) can be computed by two SQL queries. (b) Similarly, vio(D, Σcind) can be computed by a
single SQL query. (c) These SQL queries use pattern tableaux of CFD’s (CIND’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 CFD’s and CIND’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 CFD’s or CIND’s.

5.1 Encoding CFD’s and CIND’s with Data Tables

We first show the following, by extending the encoding of [10], [22]. The pattern tableaux of all CFD’s in $\Sigma_{Cfd}$ can be encoded with three data tables, and the pattern tableaux of all CIND’s in $\Sigma_{Cind}$ can be represented as four data tables, no matter how many dependencies are in the set.

Encoding CFD’s. We encode all pattern tableaux in $\Sigma_{Cfd}$ with three tables enc$L$, enc$R$, and enc$p$, where enc$L$ (resp. enc$R$) encodes the non-negation (=, $\leq$, $\geq$, $>$) patterns in LHS (resp. RHS), and enc$p$ encodes those negation (≠) patterns. More specifically, we associate a unique id cid with each CFD in $\Sigma_{Cfd}$, and let enc$L$ consist of the following attributes: (a) cid, (b) each attribute $A$ appearing in the LHS of some CFD’s in $\Sigma_{Cfd}$, and (c) its four companion attributes $A_>$, $A_>=, A_<$, 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 enc$p$ tuple to encode a pattern $A \neq c$ in a CFD, consisting of cid, att, pos, and val, encoding the CFD id, the attribute $A$, the position (‘LHS’ or ‘RHS’), and the constant $c$, respectively. Note that the arity of enc$L$ (enc$R$) is bounded by $5 \times |R_i| + 1$, where $|R_i|$ is the arity of $R_i$, and the arity of enc$p$ is 4.

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

We populate enc$L$, enc$R$, and enc$p$ as follows. For each CFD $\varphi = R(X \rightarrow Y_T) \in \Sigma_{Cfd}$, we generate a distinct cid id$\varphi$ for it, and do the following.

(1) Add a tuple $t_1$ to enc$L$ such that (a) $t_1[cid] = id_\varphi$; (b) for each $A \in X$, $t_1[A] = \neq$ if $T_p^a(\neq, A)$ is ‘\neq’; and for each non-negation predicate op, $t_1[att] = \neq$ if $T_p^a(op, A)$ is ‘\neq’; (c) we let $t_1[B] = null$ for all other attributes $B$ in enc$L$.

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

(3) For each attribute $A \in X \cup Y$ and each $\neq$ a pattern in $T_p^a$, add a tuple $t$ to enc$p$ such that $t[cid] = id_\varphi$, $t[att] = A$, $t[val] = a$, and $t[pos] = \neq$ (resp. $t[pos] = \neq$) if $A$ appears in $X$ (resp. $Y$).

Example 8: Recall from Fig. 2 CFD’s $\varphi_2$, $\varphi_3$ and $\varphi_4$ defined on relation item. The three CFD’s are encoded with the tables shown in Fig. 4: (a) enc$L$ consists of attributes: cid, sale, price, price$_c$, and price$_t$; (b) enc$R$ consists of cid, shipping, price, price$_c$, and price$_t$; those attributes in a table with only ‘null’ pattern values do not contribute to error detection, and are thus omitted; And (c) enc$p$ is empty since all these CFD’s have no negation patterns. One can easily reconstruct these CFD’s from tables enc$L$, enc$R$ and enc$p$ by collating the tuples based on cid.

Encoding CIND’s. All CIND’s in $\Sigma_{Cind}$ can be encoded with four tables enc, enc$L$, enc$R$, and enc$p$. Here enc$L$ (resp. enc$R$) and enc$p$ 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’s. We use enc to encode the INDS embedded in CIND’s, which consists of the following attributes: (1) id representing the id of a CIND, and (2) those $X$ attributes of $R_i$ and $Y$ attributes of $R_j$ appearing in some CIND’s in $\Sigma_{Cind}$. 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 CIND $\psi = R_i[A_1 \ldots A_m; X_p] \subseteq R_i[B_1 \ldots B_m; Y_p], T_p$ in $\Sigma_{Cind}$, we generate a distinct cid id$\psi$ for it, and do the following.

(1) 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 counterpart.

(2) Add tuples to enc$p$ in the same way as their CFD counterparts.

(3) Add tuple $t$ to enc such that $t[cid] = id_\psi$. 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 5 shows the coding of CIND’s $\psi_1$ and $\psi_2$ given in Fig. 3. We use state$\psi$ and state$\psi$ in enc to denote the occurrences of attribute state in item and tax, respectively. In enc$L$ and enc$R$, the attributes with only ‘null’ patterns are omitted, for the same reason as CFD’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 $\text{vio}(D, \Sigma_{Cfd})$. 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 $\text{vio}(D, \Sigma_{Cind})$. Putting these query answers together, we get $\text{vio}(D, \Sigma)$, the violation set of $D$ w.r.t. $\Sigma$.

SQL queries for CIND’s. Below we show how the SQL query $Q(i,j)$ is generated for validating CIND’s in $\Sigma_{Cind}$, which has not been studied by previous work. For the lack of space, we put the generation of detection queries for CFD’s in the supplementary material, which is an extension of the SQL techniques for CFDs and ecFDs discussed in [22] and [10], respectively.
respectively; (2) $R$ for each $X$ Here (1) enc and CIND

$$L.A_i \text{ is null or } R_i.A_k = L.A_k \text{ or } (L.A_k, L.A_i) = (',')$$ and $$(L.A_{ij} > \text{ is null or } R_i.A_j > L.A_{ij})$$ and $$(L.A_{ij} < \text{ is null or } R_i.A_j < L.A_{ij})$$ and $$(L.A_{ij} \text{ is null or } R_i.A_j \leq L.A_{ij})$$

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

$$L.cid = N.cid \text{ and } N.pos = 'LHS' \text{ and } N.att = 'A_k' \text{ and } R_i.A_k = N.val;$$

(5) $R_i.Y = N$ is defined similarly, but with $N.pos = 'RHS'$; (6) $R_i.X = R_i.Y$ represents the following: for each $A_k$ ($k \in [1, m_1]$) and each $B_i$ ($l \in [1, m_2]$), $(H.A_k, H.B_l = \text{null or } H.B_l \neq H.A_k \text{ or } R_i.A_k = R_i.B_l)$. Intuitively, (1) $R_i.X \approx L$ and $R_i.X \approx N$ ensure that the $R_i$ tuples selected match the LHS patterns of some CIND$^p$s in $\Sigma^{(i,j)}_{\text{CIND}}$, (2) $R_i.Y \approx R$ and $R_i.Y \approx N$ check the corresponding RHS patterns of these CIND$^p$s on $R_i$ tuples; (3) $R_i.X = R_i.Y$ enforces the embedded IND$^s$; (4) $L.cid = R.cid$ and $L.cid = H.cid$ assure that the LHS and RHS patterns in the same CIND are correctly collated; and (5) not exists in $Q^{(i,j)}$ ensures that the $R_i$ tuples selected violate CIND$^p$s in $\Sigma^{(i,j)}_{\text{CIND}}$.

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

$$\psi_1: \text{hcahs (zip = '3' and city = '3' \rightarrow state = '3')}$$
$$\psi_2: \text{hcahs (hid = '3' \rightarrow hname = '3' and county = '3' and addr = '3' and phone = '3')}$$. 

The SQL queries generated 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 enc$^p$, 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 CDF$^p$s.

6 Experimental Study

We next present an extensive experimental study of CDF$^p$s and CIND$^p$s. Using real-life data, we conducted two sets of experiments to evaluate the efficiency and effectiveness of CIND$^p$s and CAND$^p$s vs. their counterparts CDF$^p$s and CIND$^p$s, 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 (Hospital Compare) is a database publicly available from U.S. Department of Health & Human Services [1]. We used two tables hcahs and hcahs-state, which record the hospital level and state level ratings of the Hospital Consumer Assessment of Healthcare Providers and Systems (HCAHPS), respectively. For table hcahs, 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: mid (measure ID), mq (question), mad (answer description), map (answer percentage), mncs (number of completed surveys), msrrp (survey response rate percentage), mfn (footnote). And for table hcahs-state, it records state level measure information: state, mid, mq, and map, among other things.

We designed 6 CDF$^p$s and 3 CIND$^p$s for Hospital, shown below in an informal way for easy of understanding:

$$\psi_1: \text{hcahs (zip = '3' and city = '3' \rightarrow state = '3')}$$
$$\psi_2: \text{hcahs (hid = '3' \rightarrow hname = '3' and county = '3' and addr = '3' and phone = '3')}$$. 

$$\psi_3: \text{hcahs (hid = '3' \rightarrow msrrp = '3')}$$. 

1041-4347 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
HOSP or paper of DBLP, ranging from 0% to 9%. For easy of comparison, we deliberately dirty the tuples in hcahps of HOSP or paper of DBLP so that using the CFDPs and CINDPs 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 CFDPs and CINDPs vs. their counterparts CFDs and CINDs, respectively and together.

Exp-1: CFDPs 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 CFDPs, 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 $\phi_5$ and $\phi_6$ is higher than CFDPs $\varphi_5$ and $\varphi_6$, as they match more $I_1$ tuples, but the RHS complexity of CFDs $\varphi_5$ and $\varphi_6$ is lower than CFDPs $\varphi_5$ and $\varphi_6$ as they are easier to check violations; And (b) the LHS complexity of CFDs $\phi_4$ and $\phi_6$ is the same as CFDPs $\phi_4$ and $\phi_6$, but the RHS complexity of CFDs $\phi_4$ and $\phi_6$ is similar to CFDPs $\phi_5$ and $\phi_5$, as they are easier to check violations. As a combined result, the running time of CFDs is lower than CFDPs on HOSP, but close to CFDPs on DBLP.

Exp-1:2: CINDPs 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 CINDPs, 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 $\psi'_4$ is higher than CINDPs $\psi_5$, as they match more $I_1$ tuples, but the RHS complexity of CIND $\psi'_4$ is lower than CINDPs $\psi_5$, as they are easier to check violations; And (b) the LHS complexity of CINDs $\rho'_2$ and $\rho'_3$ is the same as CINDPs $\rho_2$ and $\rho_3$, but the RHS complexity of CINDs $\rho'_2$ and $\rho'_3$ is lower than CINDPs $\rho_2$ and $\rho_3$, as they are easier to check violations. As a combined result, the running time of CINDs is close to CINDPs on HOSP, but is lower on DBLP.

Exp-1:3: CFDPs + CINDPs vs. CFDs + CINDs. Using the same setting as Exp-1:2, we evaluated the impacts of $|I_1|$, $|I_2|$ and
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\(^p\)s and CIND\(^p\)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\(^p\)s, CINDs, CIND\(^p\)s, CFDs + CINDs or CFD\(^p\)s + CIND\(^p\)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\(^p\)s + CIND\(^p\)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\(^p\)s vs. CFDs, (b) CIND\(^p\)s vs. CINDs and (c) CFD\(^p\)s + CIND\(^p\)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\(^p\)s, CIND\(^p\)s and CFD\(^p\)s + CIND\(^p\)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 CFDs and CINDs on HOSP 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 CFDs and CINDs is comparable to their CFDs and CINDs counterparts, which is consistent with the the static analyses: CFDs and CINDs retain the same complexity as their CFDs and CINDs counterparts. (2) CFDs and CINDs are able to capture more dirty tuples than CFDs and CINDs, due to the increased expressive power.

7 CONCLUSIONS

We have proposed CFDs and CINDs, 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 CFDs and CINDs are more powerful than CFDs and CINDs for detecting errors in real-life data. In addition, the satisfiability and implication problems for CFDs and CINDs 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 CFDs and CINDs. 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., CFDs, eCFDs [10] and CFDs [13]), without increasing the complexity of static analyses. Second, we plan to develop effective algorithms for discovering CFDs and CINDs, along the same lines as [5], [28], [36]. Third, we plan to extend the methods of [8], [17] to repair data based on CFDs and CINDs, instead of using CFDs [17], traditional FDs and INs [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.
The 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/TKDE.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 MRSA in 2012. He is a recipient of the Best Paper Award for VLDB 2010 and the Best Challenge Paper Award for ICDE 2010.

Liang Duan is a PhD student in the School of Computer Science and Engineering, Beihang University, co-supervised by Prof. Jinping 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.

Chunming Hu is an associate professor at the School of Computer Science and Engineering, Beihang University. He received his PhD degree from Beihang 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 2012 to 2013. His current research interests include data management, data quality and intelligent HCI.

1041-4347 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.