Extending Conditional Dependencies with Built-in Predicates

Citation for published version:

Digital Object Identifier (DOI):
10.1109/TKDE.2015.2451632

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
IEEE Transactions on Knowledge and Data Engineering

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
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 $\text{cfd}^p$s and $\text{cfd}^{p'}$s, respectively, by specifying patterns of data values with $\neq$, $\leq$, $<$, and $\geq$ predicates. As data quality rules, $\text{cfd}^p$s and $\text{cfd}^{p'}$s are able to capture errors that commonly arise in practice but cannot be detected by CFDs and CINDs. We establish two sets of results for central technical problems associated with $\text{cfd}^p$s and $\text{cfd}^{p'}$s. (a) One concerns the satisfiability and implication problems for $\text{cfd}^p$s and $\text{cfd}^{p'}$s, taken separately or together. These are important for, e.g., deciding whether data quality rules are dirty themselves, and for removing redundant rules. We show that despite the increased expressive power, the static analyses of $\text{cfd}^p$s and $\text{cfd}^{p'}$s retain the same complexity as their CFDs and CINDs counterparts. (b) The other concerns validation of $\text{cfd}^p$s and $\text{cfd}^{p'}$s. We show that given a set $\Sigma$ of $\text{cfd}^p$s and $\text{cfd}^{p'}$s on a database $D$, a set of SQL queries can be automatically generated that, when evaluated against $D$, return all tuples in $D$ that violate some dependencies in $\Sigma$. 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 $\text{cfd}^p$s and $\text{cfd}^{p'}$s.

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 ($\equiv$). 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 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:

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

These are CFDs: (a) $\text{cfd}_1$ ensures 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 $t[sale] = \text{‘T’}$ then $t[shipping]$ 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:

• 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, hucm}@act.buaa.edu.cn.
• W. Fan is with the RCBD 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.
Contributions & Roadmap. To this end we introduce an extension of CFDs and CINDs, investigate the static analyses of these constraints, and develop effective SQL-based techniques for detecting errors based on these extensions.

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

(2) We establish the complexity bounds for the satisfiability and implication problems for CFD\(p\)s and CIND\(p\)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\(p\)s and CIND\(p\)s do not increase the complexity for reasoning about them. In particular, we show that the satisfiability and implication problems remain (a) \(\text{NP}\)-complete and \(\text{coNP}\)-complete for CFD\(p\)s, respectively, (b) in \(\text{O}(1)\)-time (constant-time) and \(\text{EXPTIME}\)-complete for CIND\(p\)s, respectively, and (c) are undecidable when CFD\(p\)s and CIND\(p\)s are taken together. These are the \textit{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\(p\)s and CIND\(p\)s (Section 5). Given a set \(\Sigma\) of CFD\(p\)s and CIND\(p\)s on a database \(D\), we automatically generate a set of SQL queries that, when evaluated on \(D\), find all tuples in \(D\) that violate some dependencies in \(\Sigma\). 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\(p\)s) and across different relations (CIND\(p\)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\(p\)s and CIND\(p\)s is comparable to their CFDs and CINDs counterparts, which is consistent with the static analyses in Section 4, and (b) CFD\(p\)s and CIND\(p\)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\(p\)s and CIND\(p\)s, separately and taken together (Section 4), and (b) an extensive experimental study of CFD\(p\)s and CIND\(p\)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

<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>Harry Potter</td>
<td>book</td>
<td>25.99</td>
<td>0</td>
<td>T</td>
<td>WA</td>
</tr>
<tr>
<td>t2</td>
<td>Snow White</td>
<td>CD</td>
<td>9.99</td>
<td>2</td>
<td>F</td>
<td>NY</td>
</tr>
<tr>
<td>t3</td>
<td>Catch-22</td>
<td>book</td>
<td>34.99</td>
<td>20</td>
<td>F</td>
<td>DL</td>
</tr>
<tr>
<td>t4</td>
<td>Sunflowers</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>PA</td>
<td>6</td>
</tr>
<tr>
<td>NY</td>
<td>4</td>
</tr>
<tr>
<td>NJ</td>
<td>3.5</td>
</tr>
</tbody>
</table>

(b) A tax rate relation


t: item \(\text{sale} = \text{F and price} \leq 20 \rightarrow \text{shipping} = 3\)

\(\text{pf}_{d1}: \text{item (sale = F and price > 20 and price} \leq 40 \rightarrow \text{shipping} = 6\)

\(\text{pf}_{d3}: \text{item (sale = F and price > 40 \rightarrow \text{shipping} = 10)\}

\(\text{pf}_{d4}: \text{item (sale = T \rightarrow \text{price} \geq 2.99 and price} < 9.99)\)

\(\text{pind}_{d1}: \text{item (state; type \neq \text{art})} \subseteq \text{tax (state; nil)\}

Here \(\text{pf}_{d2}\) states that for any item tuple, if it is not on sale and its price is in the range \([20, 40]\), then its shipment fee must be 6; similarly for \(\text{pf}_{d1}\) and \(\text{pf}_{d3}\). These dependencies extend CFDs [22] by specifying patterns of semantically related data values in terms of predicates \(\leq\), \(\geq\), and \(\neq\). Similarly, \(\text{pf}_{d4}\) assures that for any item tuple, if it is on sale, then its price must be in the range \([2.99, 9.99]\). Finally, \(\text{pind}_{d1}\) extends CINDs [30] by specifying patterns with \(\neq\): for any item tuple \(t\), if \(t(t\text{type})\) 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 \(\text{pf}_{d1}\)–\(\text{pf}_{d4}\) and \(\text{pind}_{d1}\) as data quality rules, we find that \(D_0\) of Fig. 1 is not clean. Indeed, (a) \(t_2\) violates \(\text{pf}_{d1}\): its price is less than 20, but its shipping fee is 2 rather than 3; similarly, \(t_3\) violates \(\text{pf}_{d2}\), and \(t_4\) violates \(\text{pf}_{d3}\). (b) Tuple \(t_1\) violates \(\text{pf}_{d4}\): it is on sale but its price is not in the range \([2.99, 9.99]\). (c) The database \(D_0\) also violates \(\text{pind}_{d1}\): \(t_1\) 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 \(\text{pf}_{d1}\)–\(\text{pf}_{d4}\) and \(\text{pind}_{d1}\) can be expressed as FDs or INDs [3], which do not allow constants, or as CFDs [22] or CINDs [30], which specify patterns with equality (=) only. While there have been extensions of CFDs [10], [13], [28], none of these allows dependencies to be specified with patterns on data values in terms of built-in predicates \(\neq\), \(<\), \(\leq\), \(\geq\) or \(\geq\). 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 CFDs and CINDs to capture errors commonly found in real-life data. While one can consider arbitrary extensions, it is necessary to strike a balance between their expressive power and their complexity. In particular, we want to be able to reason about data quality rules expressed as extended CFDs and CINDs. Furthermore, we want to have effective algorithms to detect inconsistencies based on these extensions.
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 CFDp’s and CINDp’s 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 CFDp’s 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 (≠, <, ≤, >, ≥), from static analyses to error detection.

Methods have been developed for discovering CFDs [14], [28], CFDp’s [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 CFDp’s and CINDp’s 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 CFDp’s, constraint-generating dependencies (CGDs) of [6] and constrained tuple-generating dependencies (CTGDs) of [32] can express CFDp’s, and CTGDs can also express CINDp’s. The increased expressive power of CTGDs comes at the price of a higher complexity: both their satisfiability and implication problems are undecidable. Built-in predicates and arbitrary constraints are supported by CGDs, for which it is not clear whether effective SQL queries can be developed to detect errors. It is worth mentioning that Theorems 2 and 6 of this work provide lower bounds for the consistency and implication analyses of CGDs, by using patterns with built-in predicates only.

Observe that constraints specifying semantics with orderings have long been recognized, such as order dependencies [27] supporting the comparison of attributes with =, <, ≤, >, ≥, matching dependencies [20] and differential dependencies [34] that support the comparison of attributes with =, ≠, <, ≤, >, ≥ for record matching. However, different from CFDs and CFDp’s, 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 CFDp’s, by extending CFDs [22] with built-in predicates (≠, <, ≤, >, ≥) in addition to equality (=).

Consider a relational schema R defined over a finite set of attributes, denoted by attr(R). For each attribute A ∈ attr(R), its domain is specified in R, denoted as 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 <, ≤, > or ≥ is defined is totally ordered.

Syntax. A CFDp ϕ on R is a pair R(X → Y, Tp), where (1) X, Y are sets of attributes in attr(R); (2) X → Y is a standard FD, referred to as the FD embedded in ϕ; and (3) Tp is a tableau with attributes in X and Y, referred to as the pattern tableau of ϕ, where for each A in X ∪ Y and each tuple tP ∈ Tp, tP[A] is either an unnamed variable ‘’ that draws values from dom(A), or ‘op’, where op is one of {=, ≠, <, ≤, >, ≥}, and ‘op’ is a constant in 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 ϕ as (X → Y, Tp) when R is clear from the context, and denote X as LHS(ϕ) and Y as RHS(ϕ), respectively.

Example 2: The dependencies cfd1–cfd3 and pfd1–pfd4 that we have seen in Example 1 can all be expressed as CFDp’s. Some of these CFDp’s are illustrated in Fig. 2, in which ϕ1 is for FD cfd2, ϕ2 is for CFD cfd3, and ϕ3 is for pfd2, and ϕ4 is for pfd4, respectively.

Semantics. Consider CFDp ϕ = R(X → Y, Tp), where Tp = {tP1, …, tPn}.

A data tuple t of R is said to match LHS(ϕ), denoted by t[X] = Tp[X], if for each tuple tP_i (i ∈ [1, k]) in Tp and each
attribute $A$ in $X$, either (a) $t_p[A]$ is the wildcard ‘.’ (which matches any value in dom($A$)), or (b) $t[A] \neq \alpha$ if $t_p[A]$ is ‘op $\alpha$’, where the operator op (equals, not equal, less than, greater than, or greater than or equal) is interpreted by its standard semantics. Similarly, the notion that $t$ matches RHS($\phi$) is defined, denoted by $t[Y] \approx T_p[Y]$.

Intuitively, each pattern tuple $t_p$, $(i \in [1, k])$ specifies a condition via $t_p[X]$, and $t[X] \approx T_p[X]$ if $t[X]$ satisfies the conjunction of all these conditions. Similarly, $t[Y] \approx T_p[Y]$ 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$^p$ $\psi$, denoted by $I \models \psi$, if for each pair of tuples $t_1, t_2$ in $I$, if $t_1[X] = t_2[X] \approx T_p[X]$, then $t_1[Y] = t_2[Y] \approx T_p[Y]$. That is, if $t_1[X]$ and $t_2[X]$ are equal and in addition, they both match the pattern tableau $T_p[X]$, then $t_1[Y]$ and $t_2[Y]$ must also be equal to each other and they both match the pattern tableau $T_p[Y]$.

Observe that $\psi$ 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 $t_1[Y] = t_2[Y]$, then (a) $t_1[Y] = t_2[Y]$, i.e., the semantics of the embedded FDs is enforced; and (b) $t_1[Y] \approx T_p[Y]$, 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 $I \models \phi$ for each CFD$^p$ $\phi$ 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 $t_3[\text{sale}] = 'F' \text{ and } t_3[\text{price}] \leq 40$, but $t_3[\text{shipping}]$ is 20 instead of 6. Note that $t_3$ matches LHS($\varphi_3$) since it satisfies the condition specified by the conjunction of the pattern tuples in $T_3$. Similarly, tuple $t_1$ violates $\varphi_4$, since $t_1[\text{sale}] = 'T' \text{ but } t_1[\text{price}] > 9.99$. 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 ‘.’ only, without constants. (2) A CFD $(X \rightarrow Y, T_p)$ with $T_p = \{t_1, \ldots, t_k\}$ can be expressed as a set of $\varphi_1, \varphi_2$ of CFD$^p$s such that for each $i \in [1, k]$, $\varphi_1 = (X \rightarrow Y, T_p[i])$, where $T_p[i]$ 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 cfd$_2$ and CFD cfd$_3$ 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 $(R_1[X_1]: X_p \subseteq R_2[Y]: Y_p, T_p)$, where (1) $X_p$ and $Y_p$ are lists of attributes in attr($R_1$) and attr($R_2$), respectively; (2) $R_1[X_1] \subseteq R_2[Y]$ is a standard IND, referred to as the IND embedded in $\psi$; and (3) $T_p$ is a tableau, defined as 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 $t_p \in T_p[A]$ is either an unnamed variable ‘.’ that draws values from dom($A$), or ‘op $\alpha$’, where op is one of $=, \neq, <, \leq, >, \geq$ and ‘$\alpha$’ 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 $\text{pind}_4$ 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$^p$ $\psi = (R_1[X_1] \subseteq R_2[Y]: Y_p, T_p)$. An instance $(I_1, I_2)$ of $(R_1, R_2)$ satisfies the CIND$^p$ $\psi$, denoted by $(I_1, I_2) \models \psi$, iff for each tuple $t_1 \in I_1$, if $t_1[X_1] \subseteq R_2[Y]$, then there exists a tuple $t_2 \in I_2$ such that $t_1[X_1] = t_2[Y]$ and $t_2[Y_p] \approx T_p[Y_p]$.

That is, if $t_1[X_1]$ matches the pattern tableau $T_p[Y_p]$, then $\psi$ assures the existence of $t_2$ such that $(1) \ t_1[X_1] = t_2[Y]$ 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 $T_p[Y_p]$, 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 $t_1[\text{type}] \neq 'art'$, but there is no tuple $t$ in tax such that $t[\text{state}] = t_1[\text{state}] = 'WA'$. 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$^p$ $(R_1[X_1] \subseteq R_2[Y]: Y_p, T_p)$ 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 $(R_1[X_1] \subseteq R_2[Y])$ can be expressed as a CIND$^p$ $(R_1[X_1]: \text{nill} \subseteq R_2[Y]: \text{nill}, T_p)$ such that $T_p$ is simply a empty set. (2) A CIND $(R_1[X_1]: X_p \subseteq R_2[Y]: Y_p, T_p)$ with $T_p = \{t_1, \ldots, t_k\}$ can be expressed as a set of $\psi_1, \psi_2, \ldots, \psi_k$ of CIND$^p$s, where for each $i \in [1, k]$, $\psi_i = (R_1[X_1]: R_2[Y]: T_p[i])$ such that $T_p[i]$ consists of the pattern tuple $t_p$ 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 $\Sigma$.
Figure 3. Example CFPDSs

The satisfiability analysis of conditional dependencies is not only of theoretical interest, but is also important in practice. Indeed, when CFPDSs 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 CFPDSs. 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 CFPDSs are a special case of CFDPSs, the same problem exists when it comes to CFPDSs.

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

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

Proposition 1: The satisfiability problem for CFPDSs is NP-complete. \( \square \)

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

We next present an NP algorithm that, given a set \( \Sigma \) of CFPDSs 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 attr(\( R \)) = \( \{ A_1, \ldots, A_m \} \) and the total number of pattern tuples in all pattern tableaux \( T_p \) of CFPDSs 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 \( \neq, <, \leq, \geq \):\n
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 \( b_{01} \neq a_1 \) 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 \( 3 \times h + 2 \). 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 \( I = \{ (\rho(t[A_i]))_1, \ldots, \rho(t[A_m])) \} \) 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. \( \square \)

It is known [22] that the satisfiability problem for CFPDSs is in PTIME when the CFPDSs 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 CFPDSs. This tells us that the increased expressive power of CFPDSs 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 CFPDSs remains NP-complete. \( \square \)

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 \( i \in [1, 3] \), \( y_{j_i} \) is either \( x_{p_i} \) or \( \overline{x_{p_i}} \), for \( p_i \in [1, m] \). Given \( \phi \), we construct a relational schema \( R \) and a set \( \Sigma \) of CFPDSs 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 \( \text{dom}(\phi) \) that contains constant \( a \). Intuitively, for each \( R \) tuple \( t \), \( t[X_1, \ldots, X_m] \) specifies a truth assignment \( \epsilon \) 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 \( \epsilon \), respectively.
2. We then construct the set \( \text{CFPS}(\Sigma) = \Sigma_0 \cup \Sigma_1 \cup \ldots \cup \Sigma_n \cup \Sigma_{n+1} \), defined as follows.
3. \( \Sigma_0 \) contains \( n+1 \) CFPDSs, which intuitively encode the relationships of the truth values between the clauses...
Let adom(A) be the active domain of A.

We construct D as follows. First, for each attribute A, define the active domain of A to be a set adom(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 adom(A) along the same lines as the one for CFDs in Proposition 1; (b) For each CIND \( R[A_1, A_2, \ldots, A_m; X_p] \subseteq \mathcal{R}_0[B_1, \ldots, B_m; Y_p], T_p \) in \( \Sigma \), let adom(B_I) = adom(A_i) \cup adom(A_j) for each \( i \in [1, m] \), and this rule is repeatedly applied until a fixpoint of adom(A) is reached for all attributes A in \( R \).

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 \( R[i_1, \ldots, i_k] \in \mathcal{R} \), we define \( D[i_1, \ldots, i_k] = adom(A_i) \times \cdots \times adom(A_k) \), where x is the Cartesian Product operation [3]. Let \( D = \{ I_1, \ldots, I_n \} \), then it is easy to verify that D is nonempty and \( D = \Sigma \).

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

Corollary 4: The satisfiability problem for CFDs and CINDs is undecidable.

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.

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.

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 \( \Sigma \not\models \varphi \).

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 \( t_1[X] = t_2[X] \neq T_p[X] \), 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 \( \Sigma \not\models \varphi \).

Assume that the attributes attr(R) = \( \{ A_1, \ldots, A_m \} \). 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 \( \rho_1 \) and \( \rho_2 \) from all attributes A 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 \( I \not\models \varphi \).
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 \( \varphi \). 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 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 CFDPs, which does not find a counterpart in [22].

**Proposition 6:** In the absence of finite domain attributes, the implication problem for CFDPs 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 \( \Sigma \models \varphi \).

We next show the reduction from the 3SAT problem to the complement problem of the implication problem for CFDPs, 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 CFDPs defined on \( R \) such that \( \varphi \) is satisfiable iff \( \Sigma \not\models \varphi \).

The relational schema \( R \) and the set \( \Sigma \) of CFDPs' are the same as the corresponding ones in Proposition 2. Moreover, \( \varphi \) is defined as \( (Z \rightarrow Z, T_p) \), where \( T_p = \{ (_- \not\models a) \} \). Intuitively, \( \varphi \) requires that for any \( R \) tuple \( t, t[Z] \not\models a \). 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 CINDPs.** We next show that CINDPs do not make their implication analysis harder, verified by extending the proof of their CINDs counterpart given in [30].

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

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

We next present the \( \text{EXPTIME} \) algorithm. Consider \( R = (R_1, \ldots, R_n) \) and \( \psi = (R_0[X; X_p] \subseteq R_0[Y; Y_p], T_p) \). 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 \( R \) 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 \( t^\prime \) such that \( t^\prime[A_i] \in \text{adom}(A_i) \) for each attribute \( A_i \). There is an edge \( e = (R_i \rightarrow t^\prime_i, R_j \rightarrow t^\prime_j) \) in \( E \) iff there exists a CINDP \( \phi = (R_0[U; U_p] \subseteq R_j[V; V_p], T^\prime_p) \) in \( \Sigma \) such that \( t_i[U_p] \times T^\prime_p[U_p], t^\prime_j[V] = t_i[U] \) and \( t^\prime_j[V] \times T^\prime_p[V_p], e \) is labeled with the CINDP \( \phi \), i.e., \( \phi \in \{ \ell \} \). Note that an edge may have multiple labels.

(2) Let \( S_u \) be the set of nodes \( R_a : t^\prime_a \) such that \( t^\prime_a[X_p] \times T^\prime_a[X_p] \), and \( S_b \) be the set of nodes \( R_b : t^\prime_b \) such that \( t^\prime_b[Y_p] \times T^\prime_b[Y_p] \), respectively.

(3) For each node \( u \) = \( R_a : t^\prime_a \) in \( S_u \), 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 \( u = R_a : t^\prime_a \), we derive another graph \( G' \) by recursively removing edges as follows. For any \( v \) in \( G_u \), if \( v \) has a child \( v' \) from which no nodes in \( R_b : t^\prime_b \) in \( S_b \) with \( t^\prime_b[X] = t^\prime_b[Y] \) 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 \( u = R_a : t^\prime_a \) such that no nodes \( R_b : t^\prime_b \) in \( S_b \) with \( t^\prime_b[X] = t^\prime_b[Y] \) 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 \( \text{EXPTIME} \): (a) The number of nodes in graph \( G \) is bounded by the maximum number of tuples in a database instance on \( R \). Let \( |\Sigma \cup \{ \psi \}| \) be the size of \( \Sigma \) and \( \psi \), and \( |R| \) be the sum of arities of all relations in \( R \). Then the number of tuples in a database instance is bounded by \( O(|\Sigma \cup \{ \psi \}|^{|R|}) \); (b) The number of nodes in sets \( S_u \) or \( S_b \) 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 \( \text{EXPTIME} \). And, hence, the problem is in \( \text{EXPTIME} \).

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 CINDPs.

**Theorem 8:** In the absence of finite domain attributes, the implication problem for CINDPs 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 \( R = (R_1, \ldots, R_n) \), we construct another database schema \( R' = (R'_1, \ldots, R'_n) \), in which each relation \( R'_i \) (\( i \in [1, n] \)) consists of infinite domain attributes only, and a set \( \Sigma' \cup \{ \psi' \} \) of CINDPs on \( R' \) such that \( \Sigma \models \psi \) iff \( \Sigma' \models \psi' \).

1041-4374 (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.
(1) We start with constructing $\mathcal{R}'$. 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 $\rho_{i,j}(b_0)$ and $\rho_{i,j}(b_{h+1})$. Note that this is always doable. For clarity, we also denote $\rho_{i,j}$ as $\rho$ when it is clear from the context.

(2) We next define $\Sigma'$ and $\psi'$ on $\mathcal{R}'$ based on the mappings defined above. For each CIND $R_a(X; A_1, \ldots, A_m)$ $\subseteq R_b(Y; B_1, \ldots, B_n), \psi$ in $\Sigma$ and each $p \in \mathcal{P}$, we define another CIND $\psi' (R'_a[X'; A'_1, \ldots, A'_m], X'_p) \subseteq R'_b[Y'; B'_1, \ldots, B'_{n-1}], \psi'$, where (a) $X'$ (resp. $Y'$) contains those attributes in $X$ (resp. $Y$) and $\rho_{i,j}(A_m)$ in $\mathcal{R}$ that are repeated in $B_m$, and (b) $X'_p$ (resp. $Y'_p$) corresponds to $X$ (resp. $Y$) when $p \in \mathcal{P}$. Finally, one can easily verify that $\Sigma' \models \psi'$ if $\Sigma \models \psi$, i.e., $\Sigma' \models \psi'$ for each CIND $\psi' \in \Sigma_{\psi'}$. Following from this, the problem is EXPTIME-hard.

Implication analysis of CFDs' and CINDs'. When CFDs' and CINDs' 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 CFDs' and CINDs' subsume FDS and INDS, respectively, from the undecidability result for FDS and INDS, the corollary below follows immediately.

Corollary 9: The implication problem for CFDs’ and CINDs’ is undecidable.

Remarks. Inference systems play an important role for the implication analyses [3]. For the inference system of CFDs and CINDs 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 \('\).

\textit{Summary.} The complexity bounds for reasoning about CFDs' and CINDs' 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, CFDs' and CINDs' do not complicate the static analyses in the general case: the satisfiability and implication problems for CFDs and CINDs have the same complexity bounds as their counterparts for CFDs and CINDs. That is, the increased expressive power of CFDs' and CINDs' comes at a price in this special case.

5 Validation of CFDs’ and CINDs’

If CFDs' and CINDs' 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 DBMSs. 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 CFDs' and CINDs' defined on $\mathcal{R}$, and a database instance $D = (I_1, \ldots, I_n)$ of $\mathcal{R}$ as input, the subset $(I'_1, \ldots, I'_n)$ of $D$ such that for each $i \in [1, n], I'_i \subseteq I_i$ and each tuple in $I'_i$ violates at least one CFD or CIND 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 CFDs' and CINDs'. The main result of the section is as follows.

Theorem 10: Given a set $\Sigma$ of CFDs' and CINDs' 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 of relations and their arities in $\mathcal{R}$, regardless of $\Sigma$. \(\square\)
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 $\Sigma_{\text{dfp}}$ can be encoded with three data tables, and the pattern tableaux of all CIND's in $\Sigma_{\text{cind}}$ can be represented as four data tables, no matter how many dependencies are in the sets.

**Encoding CFD's.** We encode all pattern tableaux in $\Sigma_{\text{dfp}}$ with three tables $\text{enc}_L$, $\text{enc}_C$, and $\text{enc}_p$, where $\text{enc}_L$ (resp. $\text{enc}_R$) encodes the non-negation ($\neq$, $\subseteq$, $\subset$, $\supset$, $\neq$) patterns in LHS (resp. RHS), and $\text{enc}_p$ encodes those negation ($\neq$) patterns. More specifically, we associate a unique id $cid$ with each CFD's in $\Sigma_{\text{dfp}}$, and let $\text{enc}_L$ consist of the following attributes: (a) $cid$, (b) each attribute $A$ appearing in the LHS of some CFD's in $\Sigma_{\text{dfp}}$, and (c) its four companion attributes $A_{\geq}$, $A_{>}$, $A_{\leq}$, and $A_{<}$. That is, for each attribute, there are five columns in $\text{enc}_L$ for each non-negation operator. Similarly, $\text{enc}_R$ is defined. We use an $\text{enc}_p$ tuple to encode a pattern $A \neq c$ in a CFD, consisting of $cid$, att, pos, and val, encoding the CFD's id, the attribute $A$, the position (‘LHS’ or ‘RHS’), and the constant $c$, respectively. Note that the arity of $\text{enc}_L (\text{enc}_R)$ is bounded by $5 * |R_i| + 1$, where $|R_i|$ is the arity of $R_i$, and the arity of $\text{enc}_p$ 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 $\varphi = R(X \rightarrow Y; T_p)$. If $\varphi$ is not satisfiable, we can simply drop it from $\Sigma$. Otherwise it is equivalent to a $\text{CFD}^p \varphi' = R(X \rightarrow Y; T'_p)$ such that for any pattern tuples $t_p, t'_p$ in $T_p$ and for any attribute $A$ in $X \cup Y$, (a) if $t_p[A]$ is op $a$ and $t'_p[A]$ is op $b$, where op is not $\neq$, then $a = b$, and (b) if $t_p[A]$ is $a$' then so is $t'_p[A]$. That is, for each non-negation op (resp. $\neq$), there is a unique constant $a$ such that $t_p[A] = 'op a' \ (\text{resp. } t_p[A] = \neq)$ 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. CFD's of this form only. Note that there are possibly multiple $t_p[A] \neq c$ patterns in $T_p^{\neq}$.

We populate $\text{enc}_L$, $\text{enc}_R$ and $\text{enc}_p$ as follows. For each CFD $\varphi = R(X \rightarrow Y; T_p)$ in $\Sigma_{\text{dfp}}$, we generate a distinct cid $id_p$ for it, and do the following.

1. Add a tuple $t_1$ to $\text{enc}_L$ such that (a) $t_1[cid] = id_p$; (b) for each $A \in X$, $t_1[A] = \neq$ if $T_p^{\neq}(A)$ is $\neq$; and for each non-negation predicate op, $t_1[A_{op}] = 'a'$. If $T_p^{op}(A)$ is $'op a'$ (c) we let $t_1[B] = \text{null}$ for all other attributes $B$ in $\text{enc}_L$.

2. Similarly add a tuple $t_2$ to $\text{enc}_R$ 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 $\text{enc}_p$ such that $t[cid] = id_p$; $t[att] = 'A'$, $t[val] = 'a'$, and $t[pos] = 'LHS'$ (resp. $t[pos] = 'RHS'$) if attribute $A$ 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) $\text{enc}_L$ consists of attributes: cid, sale, price, price, and price; (b) $\text{enc}_R$ consists of cid, shipping, price, price, and price; those attributes in a table with only ‘null’ pattern values do not contribute to error detection, and are thus omitted; And (c) $\text{enc}_p$ is empty since all these CFD's have no negation patterns. One can easily reconstruct these CFD's from tables $\text{enc}_L$, $\text{enc}_R$ and $\text{enc}_p$ by collating the tuples based on cid.

**Encoding CIND's.** All CIND's in $\Sigma_{\text{cind}}$ can be encoded with four tables $\text{enc}_c$, $\text{enc}_L$, $\text{enc}_R$, and $\text{enc}_p$. Here $\text{enc}_L$ (resp. $\text{enc}_R$) and $\text{enc}_p$ 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 $\text{enc}$ to encode the INDs embedded in CIND's, which consists of the following attributes: (1) cid 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 $\Sigma_{\text{cind}}$. 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 $\psi = \left[ R_1[A_1 \ldots A_m]; X_p \right] \subseteq \left[ R_1[B_1 \ldots B_m]; Y_p \right]$, $T_p$ in $\Sigma_{\text{cind}}$, we generate a distinct cid $id_p$ for it, and do the following.

1. Add tuples $t_1$ and $t_2$ to $\text{enc}_L$ and $\text{enc}_R$ based on attributes $X_p$ and $Y_p$, respectively, along the same lines as their CFD counterpart.

2. Add tuples to $\text{enc}_p$ in the same way as their CFD counterparts.

3. Add tuple $t$ to enc such that $t[cid] = id_p$. For each $k \in [1, m]$, 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 CIND's $\psi_1$ and $\psi_2$ given in Fig. 3. We use state$_L$ and state$_R$ in enc to denote the occurrences of attribute state in item and tax, respectively. In $\text{enc}_L$ and $\text{enc}_R$, 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 $\text{vio}(D, \Sigma_{\text{dfp}})$. 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 $\text{vio}(D, \Sigma_{\text{cind}})$. Putting these query answers together, we get $\text{vio}(D, \Sigma)$, the violation set of $D$ w.r.t. $\Sigma$.

SQL queries for CFD's. Below we show how the SQL query $Q_{i,j}$ is generated for validating CIND's in $\Sigma_{\text{cind}}$, 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 CFD's and eCFDs discussed in [22] and [10], respectively.

---

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.
respectively; (2) the sets of attributes of checking CIND
Using the coding of Fig. 5, an X
enc
Figure 4. Encoding example of CFDPS
IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. X, NO. X, DECEMBER 2014
patterns in the same CIND
R
R
corresponding RHS
R
R
where R.i.X = R.j.Y and L.C1d = R.C1d and
Here (1) X = \{A1, ..., A11\} and Y = \{B1, ..., B92\} are the sets of attributes of R1 and R2 appearing in \(\Sigma^{(i,j)}\)
\(\Sigma^{enc}\), respectively; (2) R1.X \(\cong\) L is the conjunction of
L.Ak is null or R1.Ak = L.Ak or (L.Ak = \_') and
(L.Ak\_L is null or R1.Ak > L.Ak\_L) and
(L.Ak\_R is null or R1.Ak < L.Ak\_R) and
(L.Ak\_R is null or R1.Ak \(\cong\) L.Ak\_R)
for each \(k \in [1, m_1]\); (3) Rj.Y \(\cong\) R is defined similarly for attributes in Y; (4) R1.X \(\cong\) N is a shorthand for the conjunction below, for each \(k \in [1, m_1]\):
not exists (select \* from N
where L.C1d = N.C1d and N.pos = \_’LHS’ and
N.att = \_’A’k and R1.Ak = N.val);
(5) Rj.Y \(\cong\) N is defined similarly, but with N.pos = \_’RHS’;
(6) R1.X = R1.Y represents the following: for each Ak (\(k \in [1, m_1]\)) and each B1 (\(l \in [1, m_2]\), (H.Ak is null or H.B1 is
null or H.B1 \(\neq\) H.Ak or R1.Ak = R1.B1).
Intuitively, (1) R1.X \(\cong\) L and R1.X \(\cong\) N ensure that the R1 tuples selected match the LHS patterns of some CINDPS in \(\Sigma^{(i,j)}\). (2) Rj.Y \(\cong\) R and Rj.Y \(\cong\) N check the corresponding RHS patterns of these CINDPS on Rj tuples; (3) R1.X = R1.Y enforces the embedded INDS; (4) L.C1d = R.C1d and L.C1d = H.C1d assure that the LHS and RHS patterns in the same CINDPS are correctly collated; and (5) not exists in Q(i,j) ensures that the R1 tuples selected violate CINDPS in \(\Sigma^{(i,j)}\).

Example 10: Using the coding of Fig. 5, an SQL query Q for checking CINDPS \(\psi_1\) and \(\psi_2\) of Fig. 3 is given as follows:
select R1.* from item R1, encL L, enc\_N
where (L.type is null or R1.type = L.type or L.type = \_’) and not exist (select \* from N
where N.C1d = L.C1d and N.pos = \_’LHS’ and
N.att = \_’type’ and R1.type = N.val)
and (L.state is null or R1.state = L.state or L.state = \_’) and not exist (select \* from N
where N.C1d = L.C1d and N.pos = \_’LHS’ and
N.att = \_’state’ and R1.state = N.val)
and not exist (select R2.* from tax R2, enc H, enc\_R
where (H.state is null or H.state is null or

6 EXPERIMENTAL STUDY
We next present an extensive experimental study of CFDPS and CINDPS. Using real-life data, we conducted two sets of experiments to evaluate the efficiency and effectiveness of CFDPS and CINDPS 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) HOSP (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: mid (measure ID), mq (question), mad (answer description), map (answer percentage), mncs (number of completed surveys), msrpr (survey response rate percentage), mfn (footnote). And for table hcahps-state, it records state level measure information: state, mid, mq and map, among other things.
We designed 6 CFDPSs and 3 CINDPSs for HOSP, shown below in an informal way for easy of understanding:
\(\psi_1\): hcahps (zip = \_’ and city = \_’ → state = \_’)
\(\psi_2\): hcahps (hid = \_’ → hname = \_’ and county = \_’ and addr = \_’ and phn = \_’)
\(\psi_3\): hcahps (hid = \_’ → msrpr = \_’)

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.
For comparison, we also designed the CFDs and CINDs counterparts of the above CFDs and CINDs. Here \( \varphi_1 \vdash \varphi_4 \) and \( \psi_1 \vdash \psi_2 \) are indeed CFDs and CINDs, respectively, while \( \varphi_5, \varphi_6 \) and \( \psi_3 \) are not. We hence further designed \( \varphi_5', \varphi_6' \) and \( \psi_3' \) to approximate \( \varphi_5, \varphi_6 \) and \( \psi_3 \), respectively.

\[
\begin{align*}
\varphi_5': & \text{ hcahs (mid = '̶', and mncs = 'Not Available' → mfn = 1 and mfn \leq 14)} \\
\varphi_6': & \text{ hcahs (hid = '̶', and mid = '̶', and mncs = 'Not Available' and mncs ≠ 'Fewer than 100' → map = 0 and map \leq 100)} \\
\psi_3': & \text{ hcahs (mid, state, nil) ⊆ hcahs-state (mid, state, nil)}
\end{align*}
\]

(2) DBlp is a repository of computer science publications from 1946 to 2014 [2]. We further transformed its XML format into two tables from 1946 to 2014 [2]. We collected all the data, with their representatives shown below.

\[
\begin{align*}
\phi_1: & \text{ paper (isbn = '̶', and booktitle = '̶' → publisher = '̶')} \\
\phi_2: & \text{ title = '̶', and year = '̶', and booktitle = '̶' → type = '̶')} \\
\psi_1: & \text{ paper (booktitle = 'CleanDB' → year = 2006)} \\
\phi_2: & \text{ paper (booktitle = 'VLDB' and year >= 1975 and year < 2007)} \\
\psi_5: & \text{ paper (year >= 1975 and year < 2007)} \\
\rho_1: & \text{ paper (crossref, isbn, publisher; type = 'inproceedings' and booktitle = 'CIKM-CNKM') \subseteq proceeding (key, isbn, publisher; year = 2009)} \\
\psi_3: & \text{ paper (crossref, isbn, publisher; type = 'inproceedings' and booktitle = 'ICDE') \subseteq proceeding (key, isbn, publisher; year = 1984)}
\end{align*}
\]

(3) To evaluate the impacts of \( \varphi_1 \), we fixed \( \text{noise\%} = 9\% \), and varied \( |I_1| \) from 10K to 90K for HOS, resp. (from 100K to 900K for DBlp); And (2) to evaluate the impacts of \( \text{noise\%} \), we fixed \( |I_1| = 90K \) for HOS, resp. (900K for DBlp), and varied \( \text{noise\%} \) 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 CFDs, 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 \( \varphi_5' \) and \( \varphi_6' \) is higher than CFDs \( \varphi_5 \) and \( \varphi_6 \), as they match more \( I_1 \) tuples, but the RHS complexity of CFDs \( \varphi_5' \) and \( \varphi_6' \) is lower than CFDs \( \varphi_5 \) and \( \varphi_6 \), as they are easier to check violations; And (b) the LHS complexity of CFDs \( \varphi_1 \) and \( \varphi_6 \) is the same as CFDs \( \varphi_1 \) and \( \varphi_6 \), but the RHS complexity of CFDs \( \varphi_1 \) and \( \varphi_6 \) is similar to CFDs \( \varphi_1 \) and \( \varphi_6 \), as they are easier to check violations. As a combined result, the running time of CFDs is lower than CFDs on HOS, but close to CFDs on DBlp.

**Exp-1.2: CINDs vs. CINDs.** (1) To evaluate the impacts of \( |I_2| \), we fixed \( \text{noise\%} = 9\% \), and varied \( |I_2| \) from 10K to 90K for HOS, resp. (16K for DBlp), and varied \( |I_1| \) from 10K to 90K for HOS (resp. from 100K to 900K for DBlp); (2) To evaluate the impacts of \( |I_2| \), we fixed \( \text{noise\%} = 9\% \) and \( |I_1| = 90K \) for HOS (resp. 900K for DBlp), and varied \( |I_2| \) from 1K to 1.6K for HOS (resp. from 10K to 16K for DBlp); And (3) To evaluate the impacts of \( \text{noise\%} \), we fixed \( |I_2| = 90K \) for HOS (resp. 900K for DBlp) and \( |I_2| = 1.6K \) for HOS (resp. 16K for DBlp), and varied \( \text{noise\%} \) 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 CINDs, 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 CIND \( \psi_3 \), as they match more \( I_1 \) tuples, but the RHS complexity of CIND \( \psi_3' \) is lower than CIND \( \psi_3 \), as they are easier to check violations; And (b) the LHS complexity of CINDs \( \rho_1 \) and \( \rho_3 \) is the same as CINDs \( \rho_2 \) and \( \rho_3 \), but the RHS complexity of CINDs \( \rho_1 \) and \( \rho_3 \) is lower than CINDs \( \rho_2 \) and \( \rho_3 \), as they are easier to check violations. As a combined result, the running time of CINDs is close to CINDs on HOS, but is lower on DBlp.

**Exp-1.3: CFPs + CFPs vs. CFDs + CINDs.** Using the same setting as Exp-1.2, we evaluated the impacts of \( |I_1|, |I_2| \) and 1041-1437 (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 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

**Exp-2.** Using the same setting as Exp-1.1, Exp-1.2 and Exp-1.3, respectively, we evaluated the impacts of $|I_2|$ 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

\[
\text{accuracy}(x) = \frac{\#\text{dirty tuples found by } x}{\#\text{dirty tuples found by cfd}'s + cind}'s.
\]
Figure 9. Effectiveness of detecting violations: CFD’s vs. CFDs

Figure 10. Effectiveness of detecting violations: CIND’s vs. CINDs

Figure 11. Effectiveness of detecting violations: CFD’s + CIND’s vs. CFDs + CINDs

Table 2
Summary of violation detection accuracy

<table>
<thead>
<tr>
<th>Datasets</th>
<th>varying</th>
<th>CFDs (%)</th>
<th>CFD’s (%)</th>
<th>CINDs (%)</th>
<th>CIND’s (%)</th>
<th>CFDs + CINDs (%)</th>
<th>CFD’s + CIND’s (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOSP</td>
<td></td>
<td>[74.8, 77.7]</td>
<td>[85.5, 88.8]</td>
<td>[32.1, 33.3]</td>
<td>[44.6, 48.9]</td>
<td>[74.8, 77.7]</td>
<td>100</td>
</tr>
<tr>
<td>HOSP</td>
<td>noise</td>
<td>[76.6, 77.7]</td>
<td>[87.5, 88.8]</td>
<td>[32.8, 33.3]</td>
<td>[45.5, 47.2]</td>
<td>[76.6, 77.7]</td>
<td>100</td>
</tr>
<tr>
<td>DBLP</td>
<td></td>
<td>[12.6, 17.6]</td>
<td>[62.5, 64.5]</td>
<td>[24.8, 25.4]</td>
<td>[49.6, 50.9]</td>
<td>[25.0, 29.5]</td>
<td>100</td>
</tr>
<tr>
<td>DBLP</td>
<td>noise</td>
<td>[15.3, 20.1]</td>
<td>[62.5, 64.5]</td>
<td>[24.9, 26.3]</td>
<td>[50.8, 51.1]</td>
<td>[28.0, 31.9]</td>
<td>100</td>
</tr>
</tbody>
</table>

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 CFD’s and CIND’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 CFD’s and CIND’s is comparable to their CFDs and CINDs counterparts, which is consistent with the the static analyses: CFD’s and CIND’s retain the same complexity as their CFDs and CINDs counterparts. (2) CFD’s and CIND’s are able to capture more dirty tuples than CFDs and CINDs, due to the increased expressive power.

7 CONCLUSIONS

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

One topic for future work is to develop a dependency language that is capable of expressing various extensions of CFDs (e.g., CFD’s, eCFDs [10] and CFD’s [13]), without increasing the complexity of static analyses. Second, we are to develop effective algorithms for discovering CFD’s and CIND’s along the same lines as [5], [28], [36]. Third, we plan to extend the methods of [8], [17] to repair data based on CFD’s and CIND’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.
References


