Incorporating cardinality constraints and synonym rules into conditional functional dependencies

Wenguang Chen\textsuperscript{a}, Wenfei Fan\textsuperscript{b,c,}\textsuperscript{*}, Shuai Ma\textsuperscript{b}

\textsuperscript{a} Peking University, China
\textsuperscript{b} University of Edinburgh, UK
\textsuperscript{c} Bell Laboratories, USA

\textbf{A R T I C L E I N F O}

Article history:
Received 20 December 2008
Received in revised form 22 March 2009
Accepted 23 March 2009
Communicated by J. Chomicki

Keywords:
Computational complexity
Databases
Specification languages

\textbf{A B S T R A C T}

We propose an extension of conditional functional dependencies (CFDs), denoted by CFD\textsubscript{cs}, to express cardinality constraints, domain-specific conventions, and patterns of semantically related constants in a uniform constraint formalism. We show that despite the increased expressive power, the satisfiability and implication problems for CFD\textsubscript{cs} remain NP-complete and coNP-complete, respectively, the same as their counterparts for CFDs. We also identify tractable special cases.

© 2009 Elsevier B.V. All rights reserved.

\textbf{1. Introduction}

Conditional functional dependencies (CFDs) have recently been studied for detecting inconsistencies in relational data [14]. These dependencies are an extension of functional dependencies (FDs) by enforcing patterns of semantically related data values. In contrast to traditional FDs that were developed for improving the quality of schema, CFDs aim to improve the quality of the data. That is, CFDs are to be used as data-quality rules such that errors and inconsistencies in the data can be detected as violations of these dependencies.

While CFDs are capable of capturing more errors than traditional FDs, they are not powerful enough to detect certain inconsistencies commonly found in real-life data. To illustrate this, let us consider an example.

\textbf{Example 1.1.} Consider a relation schema:

\texttt{sale(FN: string, LN: string, street: string, city: string, state: string, country: string, zip: string, item: string, type: string)},

where each tuple specifies an item of a certain type purchased by a customer. Each customer is specified by her name (FN, LN) and address (street, city, state, country, zip). An instance \( D_0 \) of the sale schema is shown in Fig. 1.

CFDs on sale data include the following:

\( \phi_1: ([\text{country}, \text{zip}] \rightarrow \text{street}, t_1^p), \text{ and } t_1^p = (\text{uk}, \_ || \_ ) \)

\( \phi_2: (\text{country} \rightarrow \text{state}, t_2^p), \text{ where } t_2^p = (\text{uk} \parallel \text{N/A}) \).

Here \( \phi_1 \) asserts that for customers in the UK, zip code uniquely determines street. It uses a tuple \( t_1^p \) to specify a pattern: \( \text{country} = \text{UK}, \text{zip} = \_ \parallel \_ \) and \( \text{street} = \_ \parallel \_ \), where \( \_ \parallel \_ \) can take an arbitrary value. It is an “FD” that is to hold on the subset of tuples that satisfies the pattern, e.g., \{\( t_1, t_3 \)\} in \( D_0 \), rather than on the entire \( D_0 \) (in the US, for example, zip does not determine street). It is not a traditional FD since it is defined with constants. Similarly, \( \phi_2 \) assures that for any address in the UK, state must be N/A (non-applicable); this is enforced by pattern tuple \( t_2^p \); country = UK and state = N/A.

When these CFDs are used as data quality rules, one can see that either \( t_1 \) or \( t_3 \) is “dirty”: they violate the
rule \( \phi_1 \). Indeed, \( t_1 \) and \( t_3 \) are about customers in the UK and they have the same zip; however, they have different streets.

A closer examination of \( D_0 \) reveals that tuple \( t_2 \) is not error-free either. Indeed, \( t_2 \) is about a transaction for a UK customer, but (a) its state is NY rather than N/A, and (b) while its zip is the same as that of \( t_1 \) and \( t_3 \), it has a street not found in \( t_1 \) or \( t_3 \). However, these violations cannot be detected by \( \phi_1 \) and \( \phi_2 \). Indeed, these CFDs are specified with the pattern country = UK, and do not apply to tuples with country = “United Kingdom”. Although UK and United Kingdom refer to the same country, they are not treated as equal by the equality operator adopted by CFDs and FDs. In other words, CFDs and FDs do not observe domain-specific abbreviations and conventions.

Another issue concerns cardinality constraints commonly found in practice, which require that the number of tuples with a certain pattern does not exceed a predefined bound. An example is that each customer is allowed to purchase at most two distinct items on sale (with type = sale). As another example, on a school database, one may want to specify that a CS student can register for at most six courses each semester. These constraints can be expressed as neither FDs nor CFDs.

These practical concerns highlight the following questions: Can one extend CFDs to express cardinality constraints and synonym rules (domain-specific abbreviations and conventions)? Can we find an extension such that it does not increase the complexity for reasoning about these dependencies? Indeed, we want a balance between the expressive power needed to deal with these issues, and the complexity for static analyses of the dependencies.

**Contributions.** We answer these questions in this paper, by providing the following.

1. We propose an extension of CFDs, denoted by CFD’s, that is able to express cardinality constraints, synonym rules and patterns of semantically related values of CFDs in a uniform constraint formalism. For example, all constraints we have seen so far can be expressed as CFD’s.

2. We establish complexity bounds for the satisfiability problem and the implication problem associated with CFD’s. The satisfiability problem is to determine whether a set \( \Sigma \) of CFD’s has a nonempty model, i.e., whether the data quality rules in \( \Sigma \) make sense. The implication problem is to decide whether a set \( \Sigma \) of CFD’s entails another CFD \( \varphi \), i.e., whether the rule \( \varphi \) is redundant given the rules in \( \Sigma \).

We show that despite the increased expressive power of CFD’s, their satisfiability and implication problems are NP-complete and coNP-complete, respectively, the same as their counterparts for CFDs [14].

3. We identify special cases where the satisfiability and implication analyses of CFD’s are in PTIME. That is, in these practical settings we are able to reason about CFD’s efficiently.

We contend that CFD’s yield a better tool than CFDs for detecting errors, without increasing the complexity of static analyses.

**Related work.** To our knowledge, no previous work has studied extensions of CFDs to capture cardinality constraints and synonym rules.

Constraint-based data cleaning was introduced in [4], which proposed to use dependencies, e.g., FDs, inclusion dependencies (INDs) and denial constraints, to detect errors in real-life data (see, e.g., [12] for a comprehensive survey). As an extension of traditional FDs, CFDs were developed in [14], which showed that the satisfiability problem and implication problem for CFDs are NP-complete and coNP-complete, respectively. There have been extensions of CFDs to support disjunction and negation [9], and ranges of values in pattern tuples [16]. These extensions address issues quite different from the focus of CFD’s, and will be further discussed in Section 5. Algorithms have been developed for discovering CFDs [11,16] and for repairing data based on CFDs [13]. There have also been a variety of extensions of FDs [6,8,19] (see [14] for a detailed discussion about the differences between these extensions and CFDs). To the best of our knowledge, no previous work has studied how to extend CFDs or FDs to express cardinality constraints, abbreviations and conventions.

Synonym rules have been studied for record matching [2,3] in the form of transformation rules. However, no previous work has studied how to express these in dependencies, or their impact on the static analyses of dependencies.

Cardinality constraints have been studied for relational data [18] to constrain the domains of attributes, and for object-oriented databases to restrict the extents of classes [10]. Numerical dependencies [17], which generalize FDs with cardinality constraints, have also been proposed for schema design. These constraints differ from CFD’s in that they cannot constrain tuples with a pattern specified in terms of constants. Query answering has been investigated for aggregate queries, FDs and denial constraints [5,7], which differ from this work in that neither these dependencies can express cardinality constraints, nor

<table>
<thead>
<tr>
<th>FN</th>
<th>LN</th>
<th>street</th>
<th>city</th>
<th>state</th>
<th>country</th>
<th>zip</th>
<th>item</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>Joe</td>
<td>Brady</td>
<td>Mayfield</td>
<td>EDI</td>
<td>N/A</td>
<td>UK</td>
<td>EH4 8LE</td>
<td>CD1</td>
</tr>
<tr>
<td>t₂</td>
<td>Mark</td>
<td>Webber</td>
<td>Crichton</td>
<td>EDI</td>
<td>NY</td>
<td>United Kingdom</td>
<td>EH4 8LE</td>
<td>CD2</td>
</tr>
<tr>
<td>t₃</td>
<td>John</td>
<td>Hull</td>
<td>Queen</td>
<td>EDI</td>
<td>N/A</td>
<td>UK</td>
<td>EH4 8LE</td>
<td>CD3</td>
</tr>
<tr>
<td>t₄</td>
<td>William</td>
<td>Smith</td>
<td>5th Ave</td>
<td>NYC</td>
<td>NY</td>
<td>US</td>
<td>10016</td>
<td>book1</td>
</tr>
<tr>
<td>t₅</td>
<td>Bill</td>
<td>Smith</td>
<td>5th Ave</td>
<td>NYC</td>
<td>NY</td>
<td>US</td>
<td>10016</td>
<td>book2</td>
</tr>
<tr>
<td>t₆</td>
<td>Bill</td>
<td>Smith</td>
<td>5th Ave</td>
<td>NYC</td>
<td>NY</td>
<td>US</td>
<td>10016</td>
<td>book3</td>
</tr>
</tbody>
</table>

**Fig. 1.** An instance of the sale relation schema.
the impact of cardinality constraints on the satisfiability and implication analyses has been considered.

**Organization.** Section 2 defines CFD’s, followed by their satisfiability and implication analyses in Sections 3 and 4, respectively. Open issues are discussed in Section 5.

2. CFD’s: An extension of CFDs

Consider a relation schema $R$ defined over a set of attributes, denoted by $\text{attr}(R)$. For each attribute $A \in \text{attr}(R)$, its domain is specified in $R$, denoted as $\text{dom}(A)$. As will be seen in Sections 3 and 4, the domains of attributes have substantial impact on the complexity of satisfiability and implication analyses of CFD’s.

**CFD’s.** A CFD $\phi$ defined on schema $R$ is a triple $R(X \rightarrow Y, t_p, c)$, where (1) $X \rightarrow Y$ is a standard FD, referred to as the FD embedded in $\phi$; (2) $t_p$ is a tuple with attributes in $X$ and $Y$, referred to as the pattern tuple of $\phi$, where for each $A \in X \cup Y$, $t_p[A]$ is either a constant ‘$a$’ in $\text{dom}(A)$, or an unnamed (yet marked) variable ‘$.'’ that draws values from $\text{dom}(A)$; and (3) $c$ is a positive integer. We refer to $\phi$ also as a conditional functional dependency.

Intuitively, $t_p$ specifies a pattern of semantically related values for $X$ and $Y$ attributes: for any tuple $t$ in an instance of $R$, if $t[X]$ has the pattern $t_p[X]$, then $t[Y]$ must observe the pattern $t_p[Y]$. Furthermore, for all those tuples $t$ such that $t[X]$ has pattern $t_p[X]$, if we group $t[Y]$ values by $t[X]$, then the number of distinct values in (i.e., the cardinality of) each group is not allowed to exceed the bound $c$. In particular, when $c = 1$, $t[X]$ uniquely determines $t[Y]$, i.e., the FD embedded in $\phi$ is enforced on those tuples having a $t_p[X]$ pattern.

If $A$ occurs in both $X$ and $Y$, we use $t_p[A\ell]$ and $t_p[A\ell]$ to indicate its occurrence in $X$ and $Y$, respectively. We separate the $X$ and $Y$ attributes in $t_p$ with ‘$\|$', and denote $X$ as LHS($\phi$) and $Y$ as RHS($\phi$). We write $\phi$ as $(X \rightarrow Y, t_p, c)$ when $R$ is clear from the context.

**Example 2.1.** CFDs $\phi_1$ and $\phi_2$ of Example 1.1 can be expressed as CFD’s below, in which $t^1_p$ and $t^2_p$ are pattern tuples given in Example 1.1:

\[ \phi_1 := ([\text{country}, \text{zip}] \rightarrow \text{street}, t^1_p, 1), \]
\[ \phi_2 := (\text{country} \rightarrow \text{state}, t^2_p, 1). \]

The cardinality constraint described in Example 1.1 can also be written as a CFD $\phi_3$: $(fd, t^3_p, 2)$, where FD $fd$ and pattern tuple $t^3_p$ are:

\[ fd : \text{fn}, \text{ln}, \text{street}, \text{city}, \text{state}, \text{country}, \text{zip}, \text{type} \rightarrow \text{item}, \]
\[ t^3_p = (\text{...} \text{...} \text{...} \text{...} \text{...} \text{sale} \| \text{...}). \]

assuring that no customer may purchase more than two distinct items with type = sale.

**Semantics of CFD’s.** To give the semantics of CFD’s, we first extend the equality relation and revise the match operator of [14].

An extension of equality. We use a finite binary relation $R_c$ to capture synonym rules. For values $a$ and $b$, $R_c(a, b)$ indicates that $a$ and $b$ refer to the same real-world entity. For example, $R_c(\text{"William"}, \text{"Bill"})$ and $R_c(\text{"United Kingdom"}, \text{"UK"})$. We assume without loss of generality that $R_c$ is symmetric: if $R_c(a, b)$ then $R_c(b, a)$. However, $R_c$ may not be transitive: from $R_c(\text{"New York State"}, \text{"NY"})$ and $R_c(\text{"NY"}, \text{"New York City"})$ it does not follow that $R_c(\text{"New York State"}, \text{"New York City"})$.

In the sequel we assume that $R_c$ is predefined, as commonly found in practice.

We define a binary operator $\equiv$ on constants such that for any values $a$ and $b$, $a \equiv b$ iff (1) $R_c(a, b)$ or $a = b$, (2) $b \equiv a$, or (3) there exists a value $c$ such that $a \equiv c$ and $b = c$. For example, “United Kingdom” $\equiv$ “UK”.

The operator $\equiv$ naturally extends to tuples: $(a_1, \ldots, a_k) \equiv (b_1, \ldots, b_k)$ iff for all $i \in [1, k]$, $a_i \equiv b_i$. Observe that given a fixed $R_c$, whether $a \equiv b$ can be decided in polynomial time.

**Matching operator.** We revise the binary operator $\bowtie$ of [14] defined on constants and ‘.’ as follows: $\eta_1 \bowtie \eta_2$ if either (a) $\eta_1$ and $\eta_2$ are constants and $\eta_1 \equiv \eta_2$, or (b) one of $\eta_1, \eta_2$ is ‘.’. The operator $\bowtie$ extends to tuples, e.g., $(a, b) \bowtie (\_, b)$ but $(a, b) \bowtie (\_, \_)$. Beyond the bound 2.

**Semantics.** Based on $\equiv$ and $\bowtie$, we now give the semantics of CFD $\phi = R(X \rightarrow Y, t_p, c)$.

An instance $D$ of schema $R$ satisfies $\phi$, denoted by $D \models \phi$, iff for each tuple $t$ in $D$, if $t[X] \bowtie t_p[X]$, then (1) $t[Y] \bowtie t_p[Y]$, and (2) $|\pi_Y(\{Rc(\phi)\}D)| \leq c$, i.e., for all tuples $t'$ in $D$ such that $t'[X] \equiv t[X]$, there exist at most c distinct $t'[Y]$ values. Here $\pi$ and $\sigma$ are the projection and selection operators in relational algebra, respectively; and $|S|$ denotes the cardinality of a set $S$ in which no two elements $a, b$ are comparable by $\equiv$.

Intuitively, $\phi$ is a constraint defined on the set of tuples $D_\phi = \{t \mid t \in D, t[X] \bowtie t_p[X]\}$ such that (a) for each $t \in D_\phi$, the pattern $t_p[Y]$ is enforced on $t[Y]$; (b) for each set of tuples in $D_\phi$ grouped by $X$ attribute values, the number of their distinct $Y$ values is bounded by the constant $c$; that is, $\phi$ expresses a cardinality constraint on the $Y$ values of those tuples grouped by $X$; and (c) synonym rules are captured by the extension $\equiv$ of the equality relation.

Note that $\phi$ is defined on the subset $D_\phi$ of $D$ identified by $t_p[X]$, rather than on the entire $D$.

An instance $D$ of $R$ satisfies a set $\Sigma$ of CFD’s, denoted by $D \models \Sigma$, if $D \models \phi$ for each $\phi$ in $\Sigma$.

**Example 2.2.** Assume that $R_c$ consists of (“United Kingdom”, “UK”) and (“William”, “Bill”). Recall instance $D_0$ of Fig. 1 and CFD’s $\phi_1, \phi_2$ and $\phi_3$ of Example 2.1. Observe the following: (a) tuple $t_2$ in $D_0$ violates $\phi_2$, since $t_2[\text{country}] \bowtie \text{UK}$ but $t_2[\text{state}] \not\equiv \text{N/A}$; (b) $t_1, t_2$ and $t_3$ violate $\phi_1$ since they are UK records with the same zip code, but they have different streets; (c) $t_4, t_5$ and $t_6$ violate $\phi_3$, since they agree on name and address (note that William $\equiv$ Bill), all have type = sale, but they have three distinct items, beyond the bound 2.
Three special cases of CFD’s are worth mentioning. (a) Traditional FDs are CFD’s in which c is 1 and the pattern tuple consists of ‘-’ only. (b) CFDs of [14] are CFD’s in which c is fixed to be 1. (c) Constant CFD’s are CFD’s in which the pattern tuples consist of constants only, i.e., they do not contain ‘-’.

3. The satisfiability analysis

A central technical problem associated with CFD’s is the satisfiability problem.

The satisfiability problem for CFD’s is to determine, given a set Σ of CFD’s on a schema R, whether or not there exists a nonempty instance D of R such that D |= Σ. The set Σ is said to be satisfiable if such an instance exists.

Intuitively, the satisfiability problem is to decide whether a set of CFD’s makes sense or not. When CFD’s are used as data quality rules, the satisfiability analysis helps us detect whether the rules are dirty themselves. Any set of FDs is satisfied by a nonempty relation. In contrast, the satisfiability problem becomes NP-complete for CFD’s [14]. Since CFD’s subsume CFD’s, the satisfiability problem for CFD’s is at least as hard as for CFDs.

Example 3.1. Consider a schema R(A, B, C), and a set Σ consisting of three CFD’s defined on R: ψ1 = (A → B), (true || b). 1, ψ2 = (A → B, (false || b), 1), and ψ3 = (C → B, (false || b), 1), where dom(A) is Boolean, and b is not an attribute. Then Σ is not satisfiable. Indeed, for any nonempty instance D of R and any tuple t in D, ψ2 requires t[B] to be b’ no matter what value t[C] is, whereas ψ1 and ψ3 force t[A] to be b no matter whether t[A] is true or false.

The intractability. Despite the increased expressive power, CFD’s do not complicate the satisfiability analysis. Indeed, the satisfiability problem for CFD’s remains in NP. The proof for the result below is an extension of Theorem 3.2 in [14], its counterpart for CFDs.

Theorem 3.1. The satisfiability problem for CFD’s is NP-complete.

Proof. It is known that the satisfiability problem is already NP-hard even for constant CFDs [14]. Since CFD’s subsume CFDs, the NP lower bound for CFDs carries over to CFD’s.

We show the upper bound by presenting an NP algorithm that, given a set Σ of CFD’s on a schema R, checks whether Σ is satisfiable. Similar to CFDs [14], CFD’s have a small model property: if there is a nonempty instance D of R such that D |= Σ, then for any t in D, (t) is an instance of R and (t) |= Σ. Thus it suffices to consider single-tuple instances (t) for deciding whether Σ is satisfiable.

Assume without loss of generality that attr(R) = {A1, ..., An}. For each i ∈ [1, n], define the active domain of Ai to be a set dom(Ai) consisting of all constants of t[Ai] for all pattern tuples t[A] in Σ, plus an extra distinct value in dom(Ai) (if there exists one). Then it is easy to verify that Σ is satisfiable iff there exists a mapping ρ that assigns a value in dom(Ai) to t[Ai] for each i ∈ [1, n] such that D = (ρ(t[A1]), ..., ρ(t[An])) and D |= Σ.

Based on these, we give the NP algorithm as follows: (a) Guess a single tuple t of R such that t[Ai] ∈ dom(Ai) for each i ∈ [1, n]. (b) Check whether (t) |= Σ. If so it returns ‘yes’, and otherwise it repeats steps (a) and (b). Note that step (b) involves checking whether x = y, which can be done in PTIME in the sizes of Σ and Rc, where Rc is the relation given in the definition of ≡. Hence the algorithm is in NP, and so is the satisfiability problem. □

A tractable case. As shown by Example 3.1, the complexity is introduced by attributes in CFD’s with a finite domain. This motivates us to consider the following special case.

A set Σ of CFD’s is said to be bounded by a constant k if at most k attributes in the CFD’s of Σ have a finite domain. In particular, when k = 0, all CFD’s in Σ are defined in terms of attributes with an infinite domain.

Bounded CFD’s make our lives much easier. Indeed, an extension of the proof of Proposition 3.5 in [14] suffices to show the following.

Proposition 3.2. It is in PTIME to determine whether a set Σ of CFD’s is satisfiable if Σ is bounded by a constant k.

Proof. When Σ is bounded by k, we develop a PTIME algorithm to determine whether Σ is satisfiable, which is based on a modified chase (see, e.g., [1] for the chase), and the small model property identified in the proof of Theorem 3.1. The algorithm is an extension of the one for CFDs (Proposition 3.5 in [14]) to further deal with finite domain attributes and the ≡ operator. Assume without loss of generality that Σ is defined on a schema R, and only attributes Ai in CFD’s of Σ have a finite domain, for i ∈ [1, k].

The algorithm checks whether there exists a tuple t of R such that t |= Σ. Initially t[Ai] is a distinct variable xA for each A ∈ attr(R). For all i ∈ [1, k] and for each value in dom(Ai) assigned to xAi, the algorithm does the following.

(a) For each CFD’s Φ = R(X → Y, t, p, c) in Σ, chase t using Φ: if t[X] t[X], then chase the t[Y] such that t[Y] t[Y] as long as t[Y] does not already contain a constant that does not match the corresponding field in t[X].

We extend the match operator ∼ to accommodate variables xB: xB ∼ xA, but xB ∼ η when η is a constant or a variable.

(b) For each attribute B ∈ attr(R), if t[B] is still xB after step (a), assign a distinct value from dom(B) to xB, which does not appear in Σ and Rc; note that dom(B) must be infinite in this case by the definition of t.

(c) If t |= Σ then return “yes”; “no” is returned if for all possible valuations to xA, for i ∈ [1, k], it cannot instantiate t such that t |= Σ.

The algorithm is in O(∥Σ∥2 |Rc|m2) time, i.e., in PTIME when k is fixed, where |Σ| is the size of Σ, |Rc| is the size of Rc (in the definition of ≡), and m is the maximum cardinality of finite domains dom(Ai) for i ∈ [1, k].

We next show that the algorithm returns “yes” if and only if Σ is satisfiable.

If the algorithm returns “yes”, there exists a tuple t such that t |= Σ. Thus Σ is satisfiable.

Conversely, if Σ is satisfiable, there exists a tuple t such t |= Σ. We show that the algorithm returns “yes”.
We consider without loss of generality satisfiable Theorem 4.2. For each attribute $A \in \text{attr}(R)$, if $t'[A]$ is a constant, then $t'[A] \equiv t[A]$. Moreover, there exist no conflicts since $t \models \Sigma$. The assignments at step (b) are irrelevant since $t'[B]$’s instantiated at that step are not constrained by pattern tuples in $\Sigma$, and thus have no impact on whether $t'$ satisfies $\Sigma$. Thus after step (b), $[t'] \models \Sigma$, and the algorithm returns "yes". □

4. The implication analysis

We next investigate another central technical problem associated with CFD’s.

Consider a set $\Sigma$ of CFD’s and a single CFD $\xi$ defined on the same schema $R$. We say that $\Sigma$ implies $\phi$, denoted by $\Sigma \models \phi$, iff for all instances $D$ of $R$, if $D \models \Sigma$ then $D \models \phi$. We consider without loss of generality satisfiable $\Sigma$ only.

The implication problem for CFD’s is to determine, given a set $\Sigma$ of CFD’s and a CFD $\xi$ defined on the same schema, whether $\Sigma \models \phi$.

The implication analysis helps us identify and eliminate redundant data quality rules.

As examples of the implication analysis, we present two simple results.

Proposition 4.1. For any CFD’s of the form:

$\phi': R(X \rightarrow Y, t_p, c), \quad \phi: R(X \rightarrow Y, t_p, c')$

(a) $\phi \models \phi'$ if $c \leq c'$; and
(b) if $\phi$ is a constant CFD, $\phi \models \phi'$ even when $c' = 1$ and $c > c'$.

Proof. (a) This can be easily verified by the definition of CFD’s. (b) We show that for any instance $D$ of $R$, if $D \models \phi$ then $D \models \phi'$. Observe that for any tuple $t \in D$, if $t[X] \equiv t_p[X]$, then $t[Y] \equiv t_p[Y]$. Hence for all tuples $t'$ in $D$, if $t'[X] \equiv t[X]$, then $t'[Y] \equiv t_p[Y]$, i.e., $|\pi_Y(\sigma_{X \leq t[X]}D)| \leq 1$. Thus $D \models \phi'$. □

The intractability. We know that the implication problem for CFD’s is coNP-complete [14]. Below we show that the upper bound remains intact for CFD’s, along the same lines as its CFD counterpart (Theorem 4.3 in [14]).

In the rest of the section we consider a set $\Sigma$ of CFD’s and a CFD $\phi = R(X \rightarrow Y, t_p, c)$ such that $c$ is bounded by a polynomial in the sizes of $\Sigma$ and $\phi$. This assumption is acceptable since in practice, $c$ is typically fairly small.

Theorem 4.2. The implication problem for CFD’s is coNP-complete.

Proof. The implication problem for constant CFDs is coNP-hard [14]. The lower bound carries over to CFD’s, which subsume CFDs.

We show that the problem is in coNP by presenting an NP algorithm for its complement, i.e., for deciding whether $\Sigma \not\models \phi$. The algorithm is based on a small model property: if $\phi = R(X \rightarrow Y, t_p, c)$ and $\Sigma \not\models \phi$, then there exists an instance $D$ of $R$ with at most $c + 1$ tuples such that $D \models \Sigma$ and $D \not\models \phi$. That is, $D$ consists of $c + 1$ tuples $t_1, \ldots, t_{c+1}$ such that for all $i, j \in [1, c + 1], t_i[X] = t_p[X]$ and $t_i[Y] = t_p[Y]$, but either there exists $l \in [1, c + 1]$ such that $t_l[X] \neq t_p[X]$ or $t_l[Y] \neq t_p[Y]$. Thus it suffices to consider instances $D$ with $c + 1$ tuples for deciding whether $\Sigma \not\models \phi$.

Assume that $\text{attr}(R) = \{A_1, \ldots, A_n\}$. For each $i \in [1, n]$, let $\text{dom}(A_i)$ be a set consisting of (a) all constants of $t_p[A_i]$ for all pattern tuples $t_p$ in $\Sigma \cup \{\phi\}$, and (b) $c + 1$ extra distinct values in $\text{dom}(A_i)$ if they exist; if $\text{dom}(A_i)$ is finite and does not have $c + 1$ extra values, let $\text{dom}(A_i)$ be $\emptyset$. Then one can verify that $\Sigma \not\models \phi$ iff there exist mappings $\rho_1, \ldots, \rho_{c+1}$ such that $\rho_i$ maps $t_p[A_i]$ to a value in $\text{dom}(A_i)$ for each $j \in [1, n]$. $D = (\rho_1(t[A_1]), \ldots, \rho_{c+1}(t[A_{c+1}]), \ldots, \rho_{c+1}(t[A_{c+1}]))$, $D \models \Sigma$ and $D \not\models \phi$.

Based on these, we give the NP algorithm as follows: (a) Guess $c + 1$ tuples $t_1, \ldots, t_{c+1}$ of $R$ such that $t_j[A_i] \in \text{dom}(A_i)$ for each $i \in [1, n]$ and $j \in [1, c + 1]$. (b) Check whether $\{t_1, \ldots, t_{c+1}\}$ satisfies $\Sigma$, but not $\phi$. If so the algorithm returns "yes", and otherwise it repeats steps (a) and (b). As argued in the proof of Theorem 3.1, step (b) can be done in PTIME in the sizes of $\Sigma$, $\phi$, and $R$. Furthermore, $c$ is bounded by a polynomial by assumption. As a result, the algorithm is in NP and thus the implication problem is in coNP. □

Special cases. Proposition 3.2 shows that for a set of CFD’s bounded by a constant $k$, the satisfiability analysis is in PTIME. This is no longer the case for the implication problem.

Theorem 4.3. It is coNP-complete to decide, given CFD’s $\Sigma$ and $\phi$, whether $\Sigma \models \phi$ when $\Sigma \cup \{\phi\}$ is bounded by a constant $k = 3$.

Proof. The problem is in coNP by Theorem 4.2. We show that it is coNP-hard by reduction from SAT to the complement of the problem (i.e., to decide whether $\Sigma \not\models \phi$), where 3SAT is NP-complete (cf. [15]). Consider an instance $\phi = C_1 \land \cdots \land C_n$ of 3SAT, where all the variables in $\phi$ are $x_1, \ldots, x_m$. $C_j$ is of the form $y_{i_1} \lor y_{i_2} \lor y_{i_3}$, moreover, for $i \in [1, 3]$, $y_{i_j}$ is either $x_{p_j}$ or $\overline{x_{p_j}}$ for $p_j \in [1, m]$; here we use $x_{p_j}$ to indicate the occurrence of a variable in literal $i$ of clause $C_j$. Given $\phi$, we construct a relation schema $R$, an empty relation $R_c$, and a set $\Sigma \cup \{\phi\}$ of CFD’s defined on $R$, such that $\phi$ is satisfiable iff $\Sigma \not\models \phi$.

1. We define schema $R(C, V_c, X, V_v, Z)$, where $\text{dom}(C) = \{1, \ldots, n\}$, $\text{dom}(V_c) = \{b_1, b_2, b_3\}$, $\text{dom}(X) = \{x_1, \ldots, x_m\}$, which is the set of variables in $\phi$, and moreover, both $\text{dom}(V_v)$ and $\text{dom}(Z)$ are integer. Intuitively, for each tuple $t \subseteq C, t[V_c], t[X], t[V_v]$ and $t[Z]$ specify a clause $C$, a truth assignment $\xi$ (one of the eight to its three variables), one of the three variables in $C$, the truth value of the variable and the truth value of $C$ determined by $\xi$.

2. Let the set $\Sigma$ of CFD’s be $\Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup \Sigma_4$.
(a) $\Sigma_1$ encodes the relationships among attributes $C$, $V_c$, $X$ and $V_v$. For each variable in a clause $C_j (1 \leq j \leq n)$ and each value $\langle b_1, b_2, b_3 \rangle$ in $\text{dom}(V_c)$, there is a CFD in $\Sigma_1$. 

Thus there are 3 ∗ 8 CFD’s for each clause Cj in Σ1, and in total, there are 24 ∗ n CFD’s in Σ1.

Each CFD for clause Cj = yj1 ∨ yj2 ∨ yj3 is of the form of R((C, Vc, X → Vp), ηp, 1) such that tρ[C] = j, tρ[Vc] = (bij, ρij, 2), and tρ[X] = xpμ, (1 ≤ i ≤ 3). The value of tρ[Vc] is decided by the value of tρ[Vp] such that tρ[Vp] = bij if yj1 = xpμ and otherwise tρ[Vp] = 1 − bij if yj1 = xpμ.

For example, if Cj = xpμ ∨ xpμ ∨ xpμ such that 1 ≤ p1, p2, p3 ≤ m, then some possible pattern tuples are (j, (0, 10), xpμ, 0), (j, (0, 10), xpμ, 0), and (j, (0, 10), xpμ, 0).

(b) Σ2 detects certain variables from appearing in clauses. For each clause Cj and each variable xi not in Cj, two CFD’s are included in Σ2: μij,1 = R((C, X → Z), (j, xj || 1), 1) and μij,1 = R((C, X → Z), (j, xj || 0), 1). Thus no tuple v satisfies t[C] = j and t[X] = xj, since otherwise μij,1 forces t[Z] = 1 and μij,2 forces t[Z] = 0. There are (m − 3) ∗ n CFD’s in Σ2.

(c) Σ3 encodes the relationship between the truth assignment Vc of clause C and its corresponding truth value Z of C. For clause Cj and each h ∈ dom(Vc), ωh = R((C, Vc, X → Z), ηp, 1) is in Σ3, where tρh[Vc] = h, tρh[Z] = 0 if h = (0, 0, 0), i.e., C is not satisfied by the corresponding truth assignment h, and tρh[Z] = 1 otherwise. In total, Σ3 consists of eight CFD’s.

(d) Σ4 includes μi = R((C → Vc, (∈ ∗ 1), 1), μj = R((C → Vc, (∈ ∗ 1), 1), ensures that for each clause C and each variable X, there is at most one truth assignment. (3) CFD ϕ is defined as R((Z → C, X), (1 ∗ ∗ 1)) = t(X) ∨ t(X) ∨ t(X). 3 ∗ n − 1. Intuitively, ϕ assures that no more than 3 ∗ n − 1 tuples in an instance of R can have truth value 1 for their clauses.

Observe that Σ consists of (m + 21) ∗ n + 10 CFD’s. Thus the reduction is in PTIME.

We also show that ϕ is satisfiable iff Σ |= ϕ. Suppose first that ϕ is satisfiable. Then there exists a truth assignment ρ that makes ϕ true. Based on ρ, we can construct an instance D of R with 3 ∗ n tuples as follows. For each clause Cj = yj1 ∨ yj2 ∨ yj3 and each variable xpμ (i ∈ [1, 3]) in Cj, we create a tuple τ, where (a) t[C] = j; (b) t[X] = xpμ; (c) t[Z] = 1; (d) t[Vc] = 1 if xpμ is assigned true by ρ, and otherwise t[Vc] = 0; (e) t[Vp] = (bij, ρij, 2) such that for each i ∈ [1, 3], bij = 1 if yj1 is assigned true by ρ, and otherwise bij = 0. That is, t[Vc] is determined by ρ to all of its three variables. Observe that D |= Σ but D |= ϕ. Hence Σ |= ϕ.

Conversely, if Σ |= ϕ, then there exists an instance D of R consisting of 3 ∗ n tuples such that D |= Σ but D |= ϕ. Observe that there exist at most n distinct values for attribute C, and each value of C can be associated with at most three distinct values of attribute X. Based on this, we define a truth assignment ρ such that ρ(X) is true if tρv,(σx=ωx,d) is [11] and ρ(X) is false otherwise. Observe that by D |= Σ, (a) tρv,(σx=ωx,d) (i ∈ [1, m]) contains exactly one element, (b) tρv,(σx=ωx,d) (i ∈ [1, n]) contains exactly one element, and (c) tρv,(σx=ωx,d) has 3 ∗ n elements. Indeed, since D |= Σ, the truth assignment ρ makes ϕ true. Thus ϕ is satisfiable.

The proof of Theorem 4.3 actually yields a stronger result. Recall that a CFD R(X → Y, tp, c) is a CFD of [14] when c = 1.

Corollary 4.4. It remains coNP-complete to decide, given a set Σ of CFD’s and a CFD ϕ, whether Σ |= ϕ when Σ ∪ {ϕ} is bounded by a constant k = 3.

Not all is lost. Below we identify two tractable special cases. It should be remarked that while the second case below can find a counterpart for CFDs (Corollary 4.4 of [14]), its proof is quite different from that of [14]. Putting this and Corollary 4.4 together, one can tell that the extension of the equality operator and the presence of cardinality constraints take their toll in the implication analysis.

Proposition 4.5. It is in PTIME to decide, given a set Σ of CFD’s and a CFD ϕ, whether Σ |= ϕ when Σ ∪ {ϕ} is bounded by a constant k = 0, i.e., all attributes in Σ or ϕ have an infinite domain.

Proof. Observe that Σ |= ϕ iff there exists a nonempty instance D of the schema R on which Σ and ϕ are defined, such that D |= Σ ∪ {¬ϕ}. Thus it suffices to develop a PTIME algorithm to check the satisfiability of Σ ∪ {¬ϕ}.

Assume that ϕ is R((X → Y, tp, c)).

(1) Since ϕ is a CFD, the proof of Theorem 4.2 tells us that Σ ∪ {¬ϕ} is satisfiable iff there exists an instance D1 of R such that D1 consists of two tuples t1 and t2. D1 = Σ, t1[X] ⊅ tp[X] and t1[X] ⊅ tp[X], but neither t1[Y] ⊅ t2[Y], nor there exists l ∈ [1, 2] such that t1[Y] ⊅ t2[Y]. In light of these, a minor extension of the PTIME algorithm given in the proof of Proposition 3.2 suffices to check whether Σ ∪ {¬ϕ} is satisfiable. Assume without loss of generality that Σ is defined on a schema R, and only attributes A1 in CFD’s of Σ have a finite domain, for i ∈ [1, k].

The algorithm checks whether there exists an instance D1 = (t1, t2) such that D1 |= Σ, but D1 ⊭ ϕ. Initially, for each attribute A ∈ X, t1[A] and t2[A] are the same distinct variable xA if tp[A] is ‘⊅’ and t1[A] = t2[A] = tp[A] if tp[A] is a constant. For each other attribute A in attr(R) (but not in X), t1[A] and t2[A] are two distinct variables xA and yA, respectively.

For all i ∈ [1, k] and for each instantiation of variables xA and yA, with values in dom(Ai), the algorithm does the following.

(a) For each CFD ϕ = R(X → Y′, tp′, c′) in Σ, chase D1 using ϕ′. If t1[X′] ⊅ tp′[X′] (i ∈ [1, 2]), then change t[Y′] such that t1[Y′] ⊅ tp′[Y′], as long as there exists no attribute A ∈ Y′ such that t1[A] is already a constant that does not match tp′[A]. Moreover, if t1[X′] ⊅ t2[X′] and c′ ≤ c, then change t[Y′] ⊅ t2[Y′] as long as there exists no attribute A ∈ Y′ such that t1[A] and t2[A] are already constants and t1[A] ≠ t2[A]. Here c = 1 since ϕ is a CFD.

(b) For each attribute B ∈ attr(R), if t1[B] (i ∈ [1, 2]) is a variable after step (a), assign a distinct value from dom(B) to t1[B]; note that dom(B) must be infinite in this case.

(c) If D1 |= Σ and D1 |= ¬ϕ, then return “yes”.
The algorithm returns "no" if for all possible valuations to $x_{A_i}$ and $y_{A_i}$ for $i \in [1, k]$, it cannot instantiate $D_1$ such that $D_1 \models \Sigma$ but $D_1 \not\models \varphi$.

From these it follows that the algorithm returns "yes" iff $\Sigma \not\models \varphi$. In addition, similar to the proof of Proposition 3.2, it is easy to see that the algorithm is in PTIME in the sizes of $\Sigma$, $\varphi$, relation $R_c$ (in the definition of $\Rightarrow$), and the maximum cardinality of the $k$ finite domains.

(2) A PTIME algorithm similar to the one given in the proof of (1) suffices to check whether $\Sigma \cup \{\lnot \varphi\}$ is satisfiable. Here the algorithm operates on $c + 1$ tuples, as described in the proof of Theorem 4.2. Since $\Sigma$ consists of CFDs only, the chase of the tuples using CFDs in $\Sigma$ is straightforward. Since all the attributes in $\Sigma$ or $\varphi$ have an infinite domain, we no longer need to check valuations to those variables denoting attributes with a finite domain. One can verify that the algorithm is in PTIME. □

5. Concluding remarks

We have proposed CFD$^c$s and shown that CFD$^c$s have the following properties. (a) CFD$^c$s are able to express CFDs of [14], cardinality constraints, and domain-specific abbreviations and conventions in a uniform constraint formalism. (b) CFD$^c$s do not complicate the static analyses: the satisfiability and implication problems for CFD$^c$s have the same complexity bounds as their counterparts for CFDs.

One topic for future work is to develop a uniform constraint language to express CFD$^c$s and other extensions of CFDs, e.g., [9,16]. Such a language, however, comes at a price of higher complexity bounds: Proposition 3.2, for example, will no longer hold. This issue deserves a full treatment. Another topic is to revise the algorithms for computing a minimum cover of a set of CFDs [14], discovering CFDs [11,16] and for repairing data based on CFDs [13], by using CFD$^c$s instead of CFDs.

Acknowledgements

We thank the anonymous reviewers for their helpful comments. Fan and Ma are supported in part by EPSRC E029213/1. Fan is a Yangtze River Scholar at Harbin Institute of Technology.

References