Conditional Functional Dependencies for Data Cleaning

Philip Bohannon[1] **Wenfei Fan**[1,2] **Floris Geerts**[2] **Xibei Jia**[2] **Anastasios Kementsietsidis**[2]

[1]Bell Laboratories
{bohannon,wenfei}@research.bell-labs.com

[2]University of Edinburgh
{wenfei,fgeerts,xjia,akements}@inf.ed.ac.uk

Abstract

We propose a class of constraints, referred to as conditional functional dependencies (CFDs)*, and study their applications in data cleaning. In contrast to traditional functional dependencies (*FDs*) that were developed mainly for schema design,* CFDs *aim at capturing the consistency of data by incorporating bindings of semantically related values. For* CFDs *we provide an inference system analogous to Armstrong's axioms for* FDs*, as well as consistency analysis. Since* CFDs *allow data bindings, a large number of individual constraints may hold on a table, complicating detection of constraint violations. We develop techniques for detecting* CFD *violations in* SQL *as well as novel techniques for checking multiple constraints in a single query. We experimentally evaluate the performance of our* CFD*-based methods for inconsistency detection. This not only yields a constraint theory for* CFDs *but is also a step toward a practical constraint-based method for improving data quality.*

1 Introduction

Recent statistics reveals that dirty date costs US businesses billions of dollars annually (cf. [6]). It is also estimated that data cleaning, a labor-intensive and complex process, accounts for 30%-80% of the development time in a data warehouse project (cf. [19]). These highlight the need for data-cleaning tools to automatically detect and effectively remove inconsistencies and errors in the data.

One of the most important questions in connection with data cleaning is how to model the consistency of the data, *i.e.,* how to specify and determine that the data is clean? This calls for appropriate application-specific integrity constraints [17] to model the fundamental semantics of the data. Unfortunately little previous work has studied this issue. Commercial ETL (extraction, transformation, loading) tools have little built-in data cleaning capability, and a significant portion of the cleaning work has still to be done manually or by low-level programs that are difficult to write and maintain [17]. A bulk of prior research has focused on the merge-purge problem (*e.g.,* [9, 11, 16, 21]) for the elimination of *approximate duplicates*, or on detecting domain discrepancies and structural conflicts (*e.g.,* [18]). There has also been recent work on *constraint repair* [2, 5, 7, 20], which specifies the consistency of data in terms of constraints, and detects inconsistencies in the data as violations of the constraints. However, previous work on constraint repair is mostly based on traditional dependencies (*e.g.,* functional and full dependencies, etc), which were developed mainly for schema design, but are often insufficient to capture the semantics of the data, as illustrated by the example below.

	CC	AC	PN	NM	STR	CT	ZIP
t_1:	01	908	1111111	Mike	Tree Ave.	NYC	07974
t_2:	01	908	1111111	Rick	Tree Ave.	NYC	07974
t_3:	01	212	2222222	Joe	Elm Str.	NYC	01202
t_4:	01	212	2222222	Jim	Elm Str.	NYC	01202
t_5:	01	215	3333333	Ben	Oak Ave.	PHI	02394
t_6:	44	131	4444444	Ian	High St.	EDI	EH4 1DT

Figure 1. An instance of the cust relation

Example 1.1: Consider the following schema, which specifies a customer in terms of the customer's phone (country code (CC), area code (AC), phone number (PN)), name (NM), and address (street (STR), city (CT), zip code (ZIP)). An instance of cust is shown in Fig. 1. Traditional functional dependencies (FDs) on a cust relation may include:

f_1: [CC, AC, PN] → [STR, CT, ZIP]
f_2: [CC, AC] → [CT]

(Recall the semantics of an FD: f_2 requires that two customer records with the same country- and area-codes also have the same city name.) Traditional FDs are to hold on all the tuples in the relation (indeed they do on Fig. 1). In contrast, the following constraint is supposed to hold only when the country code is 44. That is, for customers in the UK, ZIP determines STR:

ϕ_0: [CC = 44, ZIP] → [STR]

In other words, ϕ_0 is an FD that is to hold on the subset of tuples that satisfies the pattern "CC = 44", rather than on the entire cust relation. It is generally *not* considered an FD in the standard definition since ϕ_0 includes a *pattern* with *data values* in its specification.

The following constraints are again not considered FDs:

ϕ_1: [CC = 01, AC = 908, PN] → [STR, CT = MH, ZIP]
ϕ_2: [CC = 01, AC = 212, PN] → [STR, CT = NYC, ZIP]
ϕ_3: [CC = 01, AC = 215] → [CT =PHI]

The first constraint ϕ_1 assures that only in the US (country code 01) and for area code 908, if two tuples have the same PN, then they must have the same STR and ZIP values and furthermore, the city *must* be MH. Similarly, ϕ_2 assures that if the area code is 212 then the city must be NYC; and ϕ_3 specifies that for all tuples in the US and with area code 215, their city must be PHI (irrespective of the values of the other attributes). Observe that ϕ_1 and ϕ_2 *refine* the standard FD f_1 given above, while ϕ_3 refines the FD f_2. This refinement essentially enforces a binding of semantically related data values. Note that while tuples t_1 and t_2 in Fig. 1 do not violate f_1, they violate its refined version ϕ_1, since the city cannot be NYC if the area code is 908. □

In this example, the constraints ϕ_0, ϕ_1, ϕ_2 and ϕ_3 capture a fundamental part of the semantics of the data. However, they cannot be expressed as standard FDs and are not considered in previous work on data cleaning. Indeed, constraints that hold conditionally may arise in a number of domains. For example, an employee's pay grade may determine her title in some parts of an organization but not in others; an individual's address may determine his tax rate in some countries while in others it may depend on his salary, etc. Further, dependencies that apply conditionally appear to be particularly needed when integrating data, since dependecies that hold only in a subset of sources will hold only conditionally in the integrated data.

This paper introduces a novel extension of traditional FDs, referred to as *conditional functional dependencies* (CFDs), that are capable of capturing the notion of "correct data" in these situations. A formal framework for modelling CFDs is the first contribution of this paper (Section 2).

Our second contribution consists of techniques for reasoning about CFDs (Section 3). We show that the analysis of CFDs introduces new challenges. For example, a set of CFDs may have conflicts (*i.e.,* inconsistencies), a problem not encountered when dealing with traditional FDs. We develop techniques for determining the consistency of CFDs. We also extend Armstrong's axioms for traditional FDs (see, *e.g.,* [1]) by providing a sound and complete inference system, and give an algorithm to compute a minimal cover for a set of CFDs. These are not only useful for data cleaning as an optimization technique by minimizing the input CFDs, but also yield a CFD theory analogous to the theory of FDs.

Our third contribution is the development of SQL techniques for detecting CFD violations (Section 4). Since CFDs incorporate data values, they may in some cases be physically large, and straightforward techiques may lead to a very large number of detection queries. We develop nontrivial techniques to *merge* and efficiently check *a set* of CFDs even with a very large number of conditions. These guarantee: (a) a *single* pair of SQL queries are generated, with a bounded size independent of the data values in the CFDs, and (b) only two passes of the database are needed.

Our fourth contribution is an experimental study of the performance of our detection techniques as data size and constraint complexity vary (Section 5). We find that our techniques allow violations of CFDs with even a large number of conditions to be checked efficiently on large data sets. However, we also find that care must be taken to present the complicated **where** clauses generated by our technique to the optimizer in a way that can be easily optimized.

(a) Tableau T_1 of φ_1 = (cust:[CC, ZIP] → [STR], T_1)

CC	ZIP	STR
44	_	_

(b) Tableau T_2 of φ_2 = ([CC, AC, PN] → [STR, CT, ZIP], T_2)

CC	AC	PN	STR	CT	ZIP
_	_	_	_	_	_
01	908	_	_	MH	_
01	212	_	_	NYC	_

(c) Tableau T_3 of φ_3 = ([CC, AC] → [CT], T_3)

CC	AC	CT
_	_	_
01	215	PHI
44	141	GLA

Figure 2. Example CFDs

Finally, we present some observations, including an NP-completeness result, on constraint repair with CFDs, but defer further development to a later report (Section 6).

Our conclusion is that CFDs are a promising tool for improving data quality. We discuss related and future work in Section 7. All proofs can be found in [8].

2. Conditional Functional Dependencies

We next define CFDs. Consider a relation schema R defined over a fixed set of attributes, denoted by $\mathsf{attr}(R)$.

Syntax. A CFD φ on R is a pair $(R : X \to Y,\ T_p)$, where (1) X, Y are sets of attributes from $\mathsf{attr}(R)$, (2) $R : X \to Y$ is a standard FD, referred to as the FD *embedded in* φ; and (3) T_p is a tableau with all attributes in X and Y, referred to as the *pattern tableau* of φ, where for each A in X or Y and each tuple $t \in T_p$, $t[A]$ is either a constant 'a' in the domain *dom*(A) of A, or an unnamed variable '_'. If A appears in both X and Y, we use $t[A_L]$ and $t[A_R]$ to indicate the A field of t corresponding to A in X and Y, respectively. We write φ as $(X \to Y,\ T_p)$ when R is clear from the context.

Example 2.1: The constraints $\phi_0, f_1, \phi_1, \phi_2, f_2, \phi_3$ on cust given in Example 1.1 can be expressed as CFDs φ_1 (for ϕ_0), φ_2 (for f_1, ϕ_1 and ϕ_2, one per line, respectively) and φ_3 (for f_2, ϕ_3 and an additional [CC = 44, AC = 141] → [CT = GLA] to be used in Section 4), as shown in Fig. 2. □

If we represent both data and constraints in a uniform tableau format, then at one end of the spectrum are relational tables which are composed of data values without logic variables, and at the other end are traditional constraints which are defined in terms of logic variables but without data values, while CFDs are in the between.

Semantics. Intuitively, the pattern tableau T_p of φ refines the standard FD embedded in φ by enforcing the binding of semantically related data values. To define the semantics of φ, we first introduce a notation. For a pattern tuple t_c in T_p, we define an instantiation mapping ρ to be a mapping from t_c to a *data tuple* with no variables, such that for each attribute A in $X \cup Y$, if $t_c[A]$ is '_', ρ maps it to a constant in *dom*(A), and if $t_c[A]$ is a constant 'a', ρ maps it to the *same* value 'a'. For example, for $t_c[A, B] = (a, \_)$, one can define an instantiation ρ such that $\rho(t_c[A, B]) = (a, b)$, which maps $t_c[A]$ to itself and $t_c[B]$ to a value b in *dom*(B).

A data tuple t is said to *match* a pattern tuple t_c, denoted by $t \asymp t_c$, if there is an instantiation ρ such that $\rho(t_c) = t$. For example, $t[A, B] = (a, b) \asymp t_c[A, B] = (a, \_)$.

A relation I of R *satisfies* the CFD φ, denoted by $I \models \varphi$, if for *each pair* of tuples t_1, t_2 in the relation I, and for *each* tuple t_c in the pattern tableau T_p of φ, if $t_1[X] = t_2[X] \asymp t_c[X]$, then $t_1[Y] = t_2[Y] \asymp t_c[Y]$. That is, if $t_1[X]$ and $t_2[X]$ are equal and in addition, they both match the pattern $t_c[X]$, then $t_1[Y]$ and $t_2[Y]$ must also be equal to each other and both match the pattern $t_c[Y]$. Moreover, if Σ is a set of CFDs, we write $I \models \Sigma$ if $I \models \varphi$ for *each* CFD $\varphi \in \Sigma$.

Example 2.2: The cust relation in Fig. 1 satisfies φ_1 and φ_3 of Fig. 2. However, it does not satisfy φ_2. Indeed, tuple t_1 violates the pattern tuple $t_c = (01, 908, \_, \_, \text{MH}, \_)$ in tableau T_2 of φ_2: $t_1[\mathsf{CC}, \mathsf{AC}, \mathsf{PN}] = t_1[\mathsf{CC}, \mathsf{AC}, \mathsf{PN}] \asymp (01, 908, \_)$, but $t_1[\mathsf{STR}, \mathsf{CT}, \mathsf{ZIP}] \not\asymp (\_, \text{MH}, \_)$ since $t_1[\mathsf{CT}]$ is NYC rather than MH; similarly for t_2. □

This example tells us that while violation of a standard FD requires *two* tuples, a *single* tuple may violate a CFD.

Two special cases of CFDs are worth mentioning. First, a standard FD $X \to Y$ can be expressed as a CFD $(X \to Y,\ T_p)$ in which T_p contains a single tuple consisting of '_' only. For example, if we let T_3 of φ_3 in Fig. 2 contain only $(\_, \_, \_)$, then it is the CFD representation of the FD f_2 given in Example 1.1. Second, an instance-level FD $X \to Y$ studied in [14] is a special CFD $(X \to Y,\ T_p)$, where T_p consists of a single tuple consisting of only data values.

3 Basic Properties of CFDs

Having seen that CFDs are an extension of standard FDs, it is natural to ask whether or not we can still effectively reason about CFDs along the same lines as their FD counterpart. Is there an inference system, analogous to Armstrong's Axioms for FDs, to effectively determine whether or not a set of CFDs implies (entails) another CFD? Does a set of CFDs make sense, *i.e.*, are the CFDs consistent?

These questions are not only fundamental to CFDs, but are also important for data cleaning. Indeed, if an input set Σ of CFDs is found inconsistent, then there is *no need* to check (validate) the CFDs against the data at all. Further, it helps the user discover errors in CFD specification. When Σ is consistent, an effective implication analysis would allow us to find a *minimal cover* Σ_{mc} of Σ that is equivalent to Σ but contains no redundant CFDs, patterns or attributes; it is typically more efficient to use Σ_{mc} instead of Σ when detecting and removing inconsistencies from the data.

We answer these questions in this section. We show that as opposed to standard FDs, a set of CFDs can be inconsistent. Furthermore, the implication analysis for CFDs is more complicated than their FD counterpart. However, we show that in many practical cases the consistency of a set of CFDs can be determined efficiently. We also provide a sound and complete inference system for the implication analysis of CFDs, which is analogous to but is more involved than Armstrong's Axioms for FDs. Based on these we present a technique for computing a minimal cover of a set of CFDs.

3.1 Consistency of CFDs

One can specify any set of standard FDs, without worrying about consistency. This is no longer the case for CFDs.

Example 3.1: Consider CFD $\psi_1 = (R : [A] \to [B], T_1)$, where T_1 consists of two pattern tuples $(\_, b)$ and $(\_, c)$. Then no nonempty instance I of R can possibly satisfy ψ_1. Indeed, for any tuple t in I, while the first pattern tuple says that $t[B]$ must be b no matter what value $t[A]$ has, the second pattern requires $t[B]$ to be c.

Now assume that $dom(A)$ is bool. Consider two CFDs $\psi_2 = ([A] \to [B], T_2)$ and $\psi_3 = ([B] \to [A], T_3)$, where T_2 has two patterns (true, b_1), (false, b_2), and T_3 contains (b_1, false) and (b_2, true). While ψ_2 and ψ_3 can be separately satisfied by a nonempty instance, there exists no nonempty instance I such that $I \models \{\psi_2, \psi_3\}$. Indeed, for any tuple t in I, no matter what Boolean value $t[A]$ has, ψ_2 and ψ_3 together force $t[A]$ to take the other value from the finite domain bool. This tells us that attributes with a finite domain may complicate the consistency analysis. □

The *consistency problem for* CFDs is to determine, given a set Σ of CFDs defined on a relation schema R, whether there exists a nonempty instance I of R such that $I \models \Sigma$.

This is a nontrivial problem that is not encountered when dealing with standard FDs. It is intractable (by reduction from the non-tautology problem; see [8] for the proof), due to finite-domain attributes involved in CFDs which, as Example 3.1 shows, complicate the consistency analysis.

Theorem 3.1: *The consistency problem is* NP*-complete.* □

For data cleaning in practice, the relation schema is often fixed, and only CFDs vary and are treated as the input. To this end we develop an efficient algorithm for checking the consistency of a given set of CFDs, by generalizing the chase process for FDs (see, *e.g.*, [1] for chase). Due to the space constraint we defer the details of the algorithm to [8].

Theorem 3.2: *Given any set Σ of* CFDs *on a relation schema R, the consistency of Σ can be determined in $O(|\Sigma|^2)$ time, if either the schema R is predefined, or no attributes in Σ have a finite domain.* □

FD1: If $A \in X$, then $(X \rightarrow A,\ t_p)$, where $t_p[B]$ = ‘_’ for all $B \in X \cup \{A\}$.

FD2: If $(R\ :\ X \rightarrow A,\ t_p)$ and $B \in \mathsf{attr}(R)$, then $(R\ :\ [X, B] \rightarrow A,\ t'_p)$, where $t'_p[B]$ = ‘_’ and $t'_p[C] = t_p[C]$ for each $C \in X \cup \{A\}$.

FD3: If (1) $(X \rightarrow A_i, t_i)$ such that $t_i[X] = t_j[X]$ for all $i, j \in [1, k]$, (2) $([A_1, \ldots, A_k] \rightarrow B, t_p)$ and moreover, (3) $(t_1[A_1], \ldots, t_k[A_k]) \preceq t_p[A_1, \ldots, A_k]$, then $(X \rightarrow B, t'_p)$, where $t'_p[X] = t_1[X]$ and $t'_p[B] = t_p[B]$.

FD4: If $([B, X] \rightarrow A, t_p)$, $t_p[B]$ =‘_’, and $t_p[A]$ is a constant, then $(X \rightarrow A, t'_p)$, where $t'_p[X \cup \{A\}] = t_p[X \cup \{A\}]$.

FD5: If $([B, X] \rightarrow A, t_p)$ and $t_p[B]$ =‘_’, then $([B, X] \rightarrow A, t'_p)$, where $t'_p[C] = t_p[C]$ for each $C \in X \cup \{A\} - \{B\}$, and $t'_p[B]$ = ‘b’ for some ‘b’ $\in dom(B)$.

FD6: If $(X \rightarrow A, t_p)$ and $t_p[A] =$ ‘a’, then $(X \rightarrow A, t'_p)$, where $t'_p[A] =$ ‘_’ and $t'_p[X] = t_p[X]$.

FD7: If (1) $\Sigma \vdash_{\mathcal{I}} ([X, B] \rightarrow A, t_i)$ for $i \in [1, k]$, (2) $\mathsf{dom}(B) = \{b_1, \ldots, b_k, b_{k+1}, b_m\}$, and $(\Sigma, B = b_l)$ is not consistent except for $l \in [1, k]$, and (3) for $i, j \in [1, k]$, $t_i[X] = t_j[X]$, and $t_i[B] = b_i$, then $\Sigma \vdash_{\mathcal{I}} ([X, B] \rightarrow A, t_p)$ where $t_p[B]$ =‘_’ and $t_p[X] = t_1[X]$.

FD8: If $B \in \mathsf{attr}(R)$, $\mathsf{dom}(B) = \{b_i \,|\, i \in [1, m]\}$, and $(\Sigma, B = b_l)$ is consistent only for b_1, then $\Sigma \vdash_{\mathcal{I}} (R : B \rightarrow B, (\_, b_1))$.

Figure 3. Inference Rules for CFDs

3.2 An Inference System for CFDs

Armstrong’s Axioms for FDs are found in almost every database textbook, and are fundamental to the implication analysis of FDs. We next provide an inference system for CFDs, analogous to Armstrong’s Axioms for FDs.

The *implication problem for* CFDs is to determine, given a set Σ of CFDs and a single CFD φ on a relation schema R, whether or not Σ entails φ, denoted by $\Sigma \models \varphi$, *i.e.*, whether or not for all instances I of R, if $I \models \Sigma$ then $I \models \varphi$.

Two sets Σ_1 and Σ_2 of CFDs are *equivalent*, denoted by $\Sigma_1 \equiv \Sigma_2$, if for any instance I, $I \models \Sigma_1$ iff $I \models \Sigma_2$.

For the implication analysis of CFDs, we provide a set of inference rules, denoted by $\mathcal{I}$, in Fig. 3. To simplify the discussion we consider CFDs of the form $(R : X \rightarrow A, T_p)$, where A is a single attribute and T_p consists of a single pattern tuple t_p, written as $(R : X \rightarrow A, t_p)$. This does not lose generality since a CFD of the general form $\varphi = (R : X \rightarrow Y, T_p)$ is equivalent to a set Σ_φ of CFDs of the form above such that for each $A \in Y$ and $t_p \in T_p$, $(R : X \rightarrow A, t_p[X \cup A])$ is in Σ_φ. That is, $\Sigma_\varphi \equiv \varphi$.

Given a finite set $\Sigma \cup \{\varphi\}$ of CFDs, we use $\Sigma \vdash_{\mathcal{I}} \varphi$ to denote that φ is provable from Σ using $\mathcal{I}$.

Example 3.2: Consider a set Σ of CFDs consisting of $\psi_1 = (A \rightarrow B, (\_, b))$ and $\psi_2 = (B \rightarrow C, (\_, c))$, and a single CFD $\varphi = (A \rightarrow C, (a, \_))$, all defined on the same relation schema. Then $\Sigma \vdash_{\mathcal{I}} \varphi$ can be proved as follows:

(1) $(A \rightarrow B,\ (\_, b))$	ψ_1
(2) $(B \rightarrow C,\ (\_, c))$	ψ_2
(3) $(A \rightarrow C,\ (\_, c))$	(1), (2) and FD3
(4) $(A \rightarrow C,\ (a, c))$	(3) and FD5
(5) $(A \rightarrow C,\ (a, \_))$	(4) and FD6 □

The theorem below tells us that analogous to Armstrong’s Axioms for FDs, the inference rules of $\mathcal{I}$ characterize the implication analysis of CFDs, *i.e.*, for any set Σ of CFDs and a single CFD φ, if $\Sigma \models \varphi$ then $\Sigma \vdash_{\mathcal{I}} \varphi$ (completeness), and vice versa (soundness; see [8] for a proof).

Theorem 3.3: *The inference system $\mathcal{I}$ is sound and complete for implication of* CFDs*.* □

While the rules FD1, FD2 and FD3 in $\mathcal{I}$ are extensions of Armstrong’s Axioms for FDs, FD4—FD8 do not find a counterpart in Armstrong’s Axioms. Below we briefly illustrate the inference rules in $\mathcal{I}$.

FD1 and FD2 extend Armstrong’s Axioms reflexivity and augmentation, respectively, and are self-explanatory. A subtle issue arises when $B = A$, when B appears in both the LHS and RHS of the embedded FD $[X, B] \rightarrow A$. If so, we use $t'_p[B_L]$ = ‘_’ instead of $t'_p[B]$ = ‘_’ to refer to the B attribute in $[X, B]$ (recall from Section 2 that in a tuple t_p, the occurrences of B in the LHS and RHS can be distinguished by using $t_p[B_L]$ and $t_p[B_R]$, respectively).

FD3 extends transitivity of Armstrong’s Axioms. To cope with pattern tuples which are not found in FDs, it employs an order relation $\preceq$, defined as follows. For a pair η_1, η_2 of constants or ‘_’, we say that $\eta_1 \preceq \eta_2$ if either $\eta_1 = \eta_2 = a$ where a is a constant, or $\eta_2 =$ ‘_’. The $\preceq$ relation is naturally extended to pattern tuples. For instance, $(a, b) \preceq (\_, b)$. Intuitively, the use of $\preceq$ in FD3 assures that $(t_1[A_1], \ldots, t_k[A_k])$ is in the “scope” of $t_p[A_1, \ldots, A_k]$, *i.e.*, the pattern $t_p[A_1, \ldots, A_k]$ is applicable. In Example 3.2, FD3 can be applied because $t_1[B] = b \preceq t_2[B] = \_$, where t_1, t_2 are the pattern tuples in ψ_1, ψ_2, respectively.

FD4 tells us that for a CFD $\varphi = ([B, X] \rightarrow A, t_p)$, if $t_p[B] =$ ‘_’ and $t_p[A]$ is a constant ‘a’, then it can be simplified by dropping the B attribute from the LHS of the embedded FD. To see this, consider a relation I and *any* tuple t in I. Note that since $t_p[B] =$ ‘_’, if $t[X] \asymp t_p[X]$ then $t[B, X] \asymp t_p[B, X]$ and $t[A]$ has to be a regardless of what value $t[B]$ has. Thus φ entails $(X \rightarrow A, t_p)$.

FD5 says that in a CFD $\varphi = ([B, X] \rightarrow A, t_p)$ one can substitute a constant b for ‘_’ in $t_p[B]$. To see this, consider a relation I and any tuples t_1, t_2 in I. If $t_1[X] = t_2[X] \asymp t'_p[X]$ and moreover, $t'_p[B] = t_1[B] = t_2[B] = b$, then certainly $t_1[B, X] = t_2[B, X] \asymp t_p[B, X]$, and hence φ still applies. Thus φ implies $([B, X] \rightarrow A, t'_p)$.

FD6 tells us that in a CFD $\varphi = ([B, X] \rightarrow A, t_p)$ we can substitute ‘_’ for a constant a in $t_p[A]$. This is because, for any tuples t_1, t_2 in a relation I, if $t_1[A] = t_2[A] \asymp t_p[A] =$

a, then $t_1[A] = t_2[A] \asymp t'_p[A] =$ ‘_’. Thus $\varphi \models ([B, X] \rightarrow A, t'_p)$. Example 3.2 shows how FD6 is applied.

FD7 and FD8 deal with attributes of finite domains, which are not an issue for standard FDs since FDs have no pattern tuples. They are given *w.r.t.* a set Σ of CFDs. Specifically one needs to determine, given Σ on a relation schema R, an attribute B in $\mathsf{attr}(R)$ with a finite domain and a constant $b \in dom(B)$, whether or not there exists an instance I of R such that $I \models \Sigma$ and moreover, there is a tuple t in I such that $t[B] = b$. We say that $(\Sigma, B = b)$ is *consistent* if and only if such an instance I exists. That is, since the values of B have finitely many choices, we need to find out for which $b \in dom(B)$, Σ and $B = b$ make sense when put together. For example, consider the set $\Sigma = \{\psi_2, \psi_3\}$ given in Example 3.1, and the bool attribute A. Then neither $(\Sigma, A = \mathsf{true})$ nor $(\Sigma, A = \mathsf{false})$ is consistent.

FD7 says that for an attribute B of a finite domain and *w.r.t.* a given set Σ of CFDs, if $\Sigma \vdash_{\mathcal{I}} (X \rightarrow A, t_i)$ when $t_i[B]$ ranges over all $b \in dom(B)$ such that $(\Sigma, B = b)$ is consistent, then $t_i[B]$ can be “upgraded” to ‘_’. That is, for any instance I, if $I \models \Sigma$, then $I \models (X \rightarrow A, t_p)$, where $t_p[B] =$ ‘_’. This is because for all sensible values of $dom(B)$ that ‘_’ in $t_p[B]$ may take, $I \models (X \rightarrow A, t_i)$.

FD8 handles a special case: *w.r.t.* a given set Σ of CFDs and for an attribute B of a finite domain, if there is a *unique value* $b \in dom(B)$ such that $(\Sigma, B = b)$ is consistent, then for any instance I, if $I \models \Sigma$, we have that $t[B] = b$ for each tuple t in I; this can be expressed as a CFD $(B \rightarrow B, (\_, b))$.

From these one can see that due to the richer semantics of CFDs, $\mathcal{I}$ is more complicated than Armstrong’s Axioms. It is thus not surprising that the implication analysis of CFDs is more intriguing than their standard FD counterpart. The intractability of the theorem below is verified by reduction from the non-tautology problem to the complement of the implication problem (see [8] for the proof).

Theorem 3.4: *The implication problem for* CFD*s is co*NP-*complete.* □

The good news is that when the relation schema is predefined as commonly found in data cleaning applications, the implication analysis of CFDs can be conducted efficiently, as stated by the next theorem. The implication checking algorithm is based on a generalization of the chase process for FD implication (see [8] for the details of the algorithm).

Theorem 3.5: *Given a set Σ of* CFD*s and a single* CFD *φ defined on a schema R, whether or not $\Sigma \models \varphi$ can be decided in $O((|\Sigma| + |\varphi|)^2)$ time, if either the schema R is predefined, or no attributes in Σ have a finite domain.* □

3.3 Computing Minimal Covers of CFDs

As an application of consistency and implication analyses of CFDs, we present an algorithm for computing a minimal cover Σ_{mc} of a set Σ of CFDs. The cover Σ_{mc} is equivalent to Σ but does not contain redundancies, and thus is often smaller than Σ. Since the costs of checking and repairing CFDs are dominated by the size of the CFDs to be checked along with the size of the relational data, a non-redundant and smaller Σ_{mc} typically leads to less validating and repairing costs. Thus finding a minimal cover of input CFDs serves as an optimization strategy for data cleaning.

```
Input: A set $\Sigma$ of CFDs.
Output: A minimal cover of $\Sigma$.

1.  if $\Sigma$ is not consistent
2.  then return $\emptyset$;
3.  for each CFD $\varphi = (X \rightarrow A,\ t_p) \in \Sigma$
4.     for each attribute $B \in X$
5.        if $\Sigma \models (X - \{B\} \rightarrow A,\ (t_p[X - \{B\}], t_p(A)))$
6.        then $\Sigma := \Sigma - \{\varphi\} \cup \{(X - \{B\} \rightarrow A, (t_p[X - \{B\}],\ t_p(A)))\}$;
7.  mincover := $\Sigma$;
8.  for each CFD $\varphi = (X \rightarrow A,\ t_p) \in \Sigma$
9.        if $\Sigma - \{\varphi\} \models \varphi$
10.       then remove $\varphi$ from mincover;
11. return mincover;
```

Figure 4. Algorithm MinCover

A *minimal cover* Σ_{mc} of a set Σ of CFDs is a set of CFDs such that (1) each CFD in Σ_{mc} is of the form $(R : X \rightarrow A, t_p)$ as mentioned earlier, (2) $\Sigma_{mc} \equiv \Sigma$, (3) no proper subset of Σ_{mc} implies Σ_{mc}, and (4) for each $\varphi = (R : X \rightarrow A,\ t_p)$ in Σ_{mc}, there exists no $\varphi = (R : X' \rightarrow A,\ t_p[X' \cup A])$ in Σ_{mc} such that $X \subset X'$. Intuitively, Σ_{mc} contains no redundant CFDs, attributes or patterns.

Example 3.3: Let Σ consist of ψ_1, ψ_2 and φ given in Example 3.2. A minimal cover Σ_{mc} of Σ consists of $\psi'_1 = (\emptyset \rightarrow B, (b))$ and $\psi'_2 = (\emptyset \rightarrow C, (c))$. This is because (1) $\{\psi_1, \psi_2\} \models \varphi$ (Example 3.2), (2) ψ_1 can be simplified to ψ'_1 by removing the redundant attribute A (by the rule FD4 in $\mathcal{I}$), and (3) similarly, ψ_2 can be simplified to ψ'_2. □

We give an algorithm, MinCover, for computing a minimal cover in Fig. 4. It is an extension of its standard FD counterpart [15]. First, MinCover checks whether or not Σ is consistent (lines 1-2). If Σ is consistent, it proceeds to removes redundant attributes in the CFDs of Σ (lines 3–6). We use $(t_p[X - \{B\}], t_p(A))$ to denote the pattern tuple t'_p such that $t'_p[A] = t_p[A]$ and $t'_p[C] = t_p[C]$ for each $C \in X - \{B\}$. Next, it removes redundant CFDs from Σ (lines 8–10). From Theorems 3.2 and 3.5 it follows that MinCover is able to computes a minimal cover efficiently when the schema is predefined, in $O(|\Sigma|^3)$ time.

4 Detecting CFD Violations

A first step for data cleaning is the efficient detection of constraint violations in the data. In this section we develop techniques to detect violations of CFDs. Given an instance I of a relation schema R and a set Σ of CFDs on R, it is to find all the *inconsistent tuples* in I, *i.e.*, the tuples that (perhaps

$Q^C_{\varphi_2}$ **select** t **from** cust $t, T_2\ t_p$
where t[CC] $\asymp t_p$[CC] AND t[AC] $\asymp t_p$[AC] AND
t[PN] $\asymp t_p$[PN] AND
(t[STR] $\not\asymp t_p$[STR] OR t[CT] $\not\asymp t_p$[CT] OR t[ZIP] $\not\asymp t_p$[ZIP])

$Q^V_{\varphi_2}$ **select distinct** t[CC], t[AC], t[PN] **from** cust $t, T_2\ t_p$
where t[CC] $\asymp t_p$[CC] AND t[AC] $\asymp t_p$[AC] AND t[PN] $\asymp t_p$[PN]
group by t[CC], t[AC], t[PN]
having count(**distinct** t[STR], t[CT], t[ZIP])> 1

Figure 5. SQL queries for checking CFD φ_2

together with other tuples in I) violate some CFD in Σ. We first provide an SQL technique for finding violations of a single CFD, and then generalize it to validate multiple CFDs.

4.1 Checking a Single CFD with SQL

Consider a CFD $\varphi = (X \rightarrow Y,\ T_p)$. The following two SQL queries suffice to find the tuples that violate φ:

Q^C_{φ} **select** t **from** $R\ t,\ T_p\ t_p$
where $t[X_1] \asymp t_p[X_1]$ AND . . . AND $t[X_n] \asymp t_p[X_n]$ AND
$(t[Y_1] \not\asymp t_p[Y_1]$ OR . . . OR $t[Y_n] \not\asymp t_p[Y_n])$

Q^V_{φ} **select distinct** $t.X$ **from** $R\ t,\ T_p\ t_p$
where $t[X_1] \asymp t_p[X_1]$ AND . . . AND $t[X_n] \asymp t_p[X_n]$
group by $t.X$ **having** count(**distinct** Y)> 1

where X_i (resp. Y_j) ranges over attributes in X (resp. Y); $t[X_i] \asymp t_p[X_i]$ is a short-hand for the SQL expression ($t[X_i]$ = $t_p[X_i]$ OR $t_p[X_i]$ = '_'), while $t[Y_j] \not\asymp t_p[Y_j]$ is a short-hand for ($t[Y_j] \neq t_p[Y_j]$ AND $t_p[Y_j] \neq$ '_').

Intuitively, detection is a two-step process, each conducted by a query. Initially, query Q^C_{φ} detects *single-tuple* violations, *i.e.,* the tuples t in I that match some pattern tuple $t_p \in T_p$ on the X attributes, but t does not match t_p in the Y attributes due to a *constant* value $t_p[Y_i]$ different from value $t[Y_i]$. That is, Q^C_{φ} finds inconsistent tuples based on differences in the constants in the tuples and T_p patterns.

On the other hand, query Q^V_{φ} finds *multi-tuple* violations, *i.e.,* tuples t in I for which there exists a tuple t' in I such that $t[X] = t'[X]$ and moreover, both t and t' match a pattern t_p on the X attributes, but $t[Y_j] \neq t'[Y_j]$ for some attribute Y_j in Y. Query Q^V_{φ} uses the **group by** clause to group tuples with the same value on X and it counts the number of distinct instantiations in Y. If there is more than one instantiation, then there is a violation. It catches both tuples t and t' mentioned above as violations, although it is possible that both *pass* the test of query Q^C_{φ}.

To be precise, Q^V_{φ} returns only the X attributes of inconsistent tuples (this is caused by the **group by** . However, this has the advantage that the ouput is more concise than when we would return the complete tuples. Moreover, the complete tuples can be easily obtained from the result of the two queries by means of a simple SQL query.

Example 4.1: Recall CFD φ_2 given in Fig. 2. Over a cust instance I, the SQL queries $Q^C_{\varphi_2}$ and $Q^V_{\varphi_2}$ shown in Fig. 5 determine whether or not I satisfies φ_2. Executing these queries over the instance of Fig. 1, it returns tuples t_1, t_2 (due to $Q^C_{\varphi_2}$), and t_3 and t_4 (due to $Q^V_{\varphi_2}$). □

φ_4 = (cust:[CC, AC, PN] $\rightarrow$ [STR, CT, ZIP], T_4), where T_4 is

CC	AC	PN	STR	CT	ZIP
_	_	_	_	_	_
01	908	_	_	MH	_
01	212	_	_	NYC	_
_	_	@	@	_	@
01	215	@	@	PHI	@

Figure 6. Merged φ_2 and φ_3 CFDs

A salient feature of our SQL translation is that tableau T_p is treated an ordinary data table. Therefore, each query is bounded by the size of the embedded FD $X \rightarrow Y$ in the CFD, and is *independent* of the size (and contents) of the (possibly large) tableau T_p.

4.2 Validating Multiple CFDs

A naive way to validate a set Σ of CFDs is to use one query pair for each CFD φ in Σ. This approach requires $2 \times |\Sigma|$ passes of the underlying relation. We next present an alternative approach that only requires two passes. The key idea is to generate a *single* query pair to check all the constrains in Σ. The proposed solution works in two phases. In its first phase, it performs a linear scan of all the tableaux belonging to CFDs in Σ and *merges* them, generating a single tableau called T_Σ. Intuitively, tableau T_Σ is such that it captures the constraints expressed by all the tableaux of the CFDs in Σ. Then, in its second phase, it generates a query pair that finds inconsistent tuples violating CFDs in Σ.

4.2.1 Merging Multiple CFDs

Consider a set Σ which, *w.l.o.g.*, contains just two CFDs φ and φ' on R, where $\varphi = (X \rightarrow Y,\ T)$ and $\varphi' = (X' \rightarrow Y',\ T')$. There are two main challenges for the generation of the merged tableau T_Σ. The first challenge is that tableaux T and T' may not be *union-compatible*, *i.e.,* $X \neq X'$ or $Y \neq Y'$. We thus need to extend tableau T (resp. T') with all the attributes in $Z = (X \cup Y) - (X' \cup Y')$ (resp. $(X' \cup Y') - (X \cup Y)$ for T'). For each attribute A in Z and each tuple t_c in the original tableau T, we set the value of $t_c[A]$ to be a *special symbol* denoted by '@', which denotes intuitively a *don't care* value. After this extension, the resulted tableaux are union-compatible. Then, tableau T_Σ is defined to be their union. Figure 6 shows how the CFDs φ_2 and φ_3 of Fig. 2 can be made union-compatible.

Given the presence of "@", we need to reformulate CFD satisfaction. Consider a tuple $t_c[X, Y]$ in a tableau that includes '@'. We use $X^{free}_{t_c}$ and $Y^{free}_{t_c}$ to denote the subset of X and Y attributes of t_c that is '@'-free, *i.e.,* it has no '@' symbol. A relation I of R *satisfies* the CFD φ, denoted by $I \models \varphi$, if for *each pair* of tuples t_1, t_2 in the relation I, and for *each* tuple t_c in the pattern tableau T_p of φ, if $t_1[X^{free}_{t_c}] = t_2[X^{free}_{t_c}] \asymp t_c[X^{free}_{t_c}]$, then $t_1[Y^{free}_{t_c}] = t_2[Y^{free}_{t_c}] \asymp t_c[Y^{free}_{t_c}]$.

For the second challenge, consider the translation of a single CFD into an SQL query pair. Note that the translation assumes implicit knowledge of which attributes are in the

id	CC	AC	CT
1	_	_	@
2	01	215	@
3	44	141	@
4	@	@	_

(a) Tableau T_Σ^X

id	CT	AC
1	_	@
2	PHI	@
3	GLA	@
4	@	_

(b) Tableau T_Σ^Y

Figure 7. T_Σ for CFDs φ_3 and φ_5

X and Y sets and treats the translation of each attribute set differently. Now, consider two simple CFDs on R, namely, $\varphi = (A \rightarrow B, T)$ and $\varphi' = (B \rightarrow A, T')$. Suppose that we have made the tableaux of the CFDs union-compatible. One might want to take the union of these two tableaux to generate T_Σ. How can we translate tableau T_Σ into an SQL query pair? Clearly, we cannot directly use the translation given earlier since we do not know how to treat the join of an attribute like, say, A. Attribute A is in X for tuples coming from φ, while it is part of Y for tuples coming from φ'. Thus we need to distinguish the two sets of tuples and treat the translation of each set separately.

We accommodate this by splitting the tableau T of each CFD $\varphi = (R : X \rightarrow Y,\ T)$ into two parts, namely, T^X and T^Y, one tableau for the X and one for Y attributes of φ. Then, tableaux T_Σ^X (and similarly T_Σ^Y) is generated by making all the T^X tableaux in Σ union-compatible. Note that an attribute can appear in both T_Σ^X and T_Σ^Y. To be able to restore pattern tuples from T_Σ^X and T_Σ^Y, we create a distinct *tuple id* $t.id$ for each pattern tuple t, and associates it with the corresponding tuples in T_Σ^X and T_Σ^Y. For example, consider CFD φ_3 shown in Fig. 2 and $\varphi_5 = (\text{cust} : [\text{CT}] \rightarrow [\text{AC}], T_5)$, where T_5 consists of a single tuple $(\_, \_)$. Figure 7 shows their merged T_Σ^X and T_Σ^Y tableaux. Note that attributes CT and AC appear in both tableaux.

4.2.2 Query Generation

During the second phase of our approach, we translate tableau T_Σ into a single SQL query pair. This translation, however, introduces new challenges. Recall that query Q_φ^V, for some CFD $\varphi = (R : X \rightarrow Y,\ T)$, requires a **group by** clause over all the X attributes. Now, consider tableau T_Σ^X in Fig. 7. It is not hard to see that if we use the **group by** clause over all the attributes in T_Σ^X, we are not going to detect all (if any) inconsistencies since, for example, for the first three tuples in T_Σ^X the '@' in attribute CT indicates that, while detecting inconsistencies, we should only group by the first two attributes and ignore the value of attribute CT. Similarly for the last tuple in T_Σ^X, the '@' in attributes CC and AC indicates that while detecting inconsistencies for these tuples, we should only consider the value of CT. The example suggests that our SQL query should change the set of **group by** attributes, based on the contents of each tuple. In what follows, we show how this can be achieved while still keeping the query size bounded by the size of the embedded FD $X \rightarrow Y$ and *independent* of the size of the tableau. Central to our approach is the use of the **case** clause of SQL (supported by commercial DBMS like DB2).

Consider the merged tableaux T_Σ^X and T_Σ^Y from a set Σ of CFDs over a relation schema R and let I be an instance of R. Then, the following two SQL queries can be used to detect inconsistent tuples of I violating φ:

```
$Q_\Sigma^C$   select $t$   from   $R\ t,\ T_\Sigma^X\ t_p^X,\ T_\Sigma^Y\ t_p^Y$
       where$t_p^X$.id = $t_p^Y$.id AND
              $t[X_1] \asymp t_p^X[X_1]$ AND . . . $t[X_n] \asymp t_p^X[X_n]$ AND
              $(t[Y_1] \not\asymp t_p^Y[Y_1]$ OR . . . $t[Y_n] \not\asymp t_p^Y[Y_n])$

$Q_\Sigma^V$   select distinct   $t^M.X$   from  Macro $t^M$
       group by $t^M.X$
       having count(distinct Y)$> 1$
```

where Macro is:

```
select (case $t_p^X[X_i]$ when "@" then "@" else $t[X_i]$ end )AS $X_i$ . . .
       (case $t_p^Y[Y_j]$ when "@" then "@" else $t[Y_j]$ end )AS $Y_j$ . . .
from  $R\ t, T_\Sigma^X\ t_p^X, T_\Sigma^Y\ t_p^Y$
where$t_p^X$.id = $t_p^Y$.id AND
       $t[X_1] \asymp t_p^X[X_1]$ AND . . .  AND $t[X_n] \asymp t_p^X[X_n]$
```

where $t[X_i] \asymp t_p[X_i]$ now accounts for the '@' and is a short-hand for ($t[X_i]$ = $t_p[X_i]$ OR $t_p[X_i]$ = '_' OR $t_p[X_i]$ = '@'), while $t[Y_j] \not\asymp t_p[Y_j]$ is a short-hand for ($t[Y_j] \neq t_p[Y_j]$ AND $t_p[Y_j] \neq$ '_' AND $t_p[Y_j] \neq$ '@').

More specifically, query Q_Σ^C is similar in spirit to the SQL query that checks for inconsistencies of constants between the relation and the tableau, for a single CFD. The only difference is that now the query has to account for the presence of the '@ 'symbol in the tableau. Now, we turn our attention to relation Macro which is of the same sort as T_Σ^X and T_Σ^Y (we rename attributes that appear in both tableaux so as not to appear twice). Relation Marco is essentially the join on X of relation I with the result of the join on tuple id $t.id$ of the two tableaux. The value of each attribute, for each tuple t^M in Marco, is determined by the **case** clause. In more detail, $t^M[X_i]$ is set to be '@' if $t_p^X[X_i]$ is '@', and is $t[X_i]$ otherwise; similarly for $t^M[Y_j]$. Note that relation I is not joined on Y with the tableaux. Thus if for some tuple t with $t[X] \asymp t_p^X[X]$, there exists an attribute Y_j with $t_p^Y[Y_j]$ a constant and $t[Y_j] \neq t_p^Y[Y_j]$ (*i.e.*, t is inconsistent *w.r.t.* t_p) then $t^M[Y_j]$ is set to be $t[Y_j]$. This creates no problems since this inconsistent tuple is already detected by Q_Σ^C.

Intuitively, Macro considers each tuple in the tableau, and uses it as a *mask* over the tuples of the relation. If the tableau tuple indicates a *don't care* value for an attribute, all the (possibly different) attribute values in the relation tuples are masked and replaced by an '@' in Macro. Figure 8 shows the result of joining the fourth tuple of tableaux T_Σ^X and T_Σ^Y in Fig. 7 with the cust relation of Fig. 1. Note that the query masks the attributes values of CC and AC. This masking allows the subsequent **group by** over X to essentially consider, for each tuple, only the subset of X that does not have any *don't care* values. Note that although $X = \{\text{CC}, \text{AC}, \text{CT}\}$, the **group by** by query Q_Σ^V essentially performs a **group by** over *only* attribute CT. The query returns the NYC tuples which violate φ_5.

CC	AC	CT	CT′	AC′
@	@	NYC	@	908
@	@	NYC	@	212
@	@	PHI	@	215
@	@	EDI	@	131

Figure 8. Marco relation instance

In this way we generate a *single pair* of SQL queries to validate a *set* Σ of CFDs, while guaranteeing that the queries are bounded by the size of the embedded FDs in Σ, *independent* of the size of the tableaux in Σ. Furthermore, to validate Σ only two passes of the database is required.

5 Experimental Study

In this section, we present our findings about the performance of our schemes for detecting CFD violations over a variety of data sizes, and number and complexity of CFDs.

Setup: For the experiments, we used DB2 on an Apple Xserve with 2.3GHz PowerPC dual CPU and 4GB of RAM.

Data: Our experiments used an extension of the relation in Fig. 1. Specifically, the relation used models individual's tax-records and includes 8 additional attributes, namely, the state ST a person resides in, her marital status MR, whether she has dependents CH, her salary SA, tax rate TX on her salary, and 3 attributes recording tax exemptions, based on marital status and the existence of dependents.

To populate the relation we collected real-life data: the zip and area codes for major cities and towns for all US states. Further, we collected the tax rates, tax and income brackets, and exemptions for each state. Using these data, we wrote a program that generates synthetic tax records.

We vary two parameters of the data instance in our experiments, denoted by SZ and NOISE. SZ determines the tuple number in the tax-records relation and NOISE the percentage of dirty tuples. As the data is generated, with probability NOISE, an attribute on the RHS of a CFD is changed from a correct to incorrect value (*e.g.*, a tax record for a NYC resident with a Chicago area code).

CFDs: We used CFDs that represent real-world constraints such as (a) zip codes determine states, (b) zip codes and cities determine states (a city by itself does not suffice since many states have cities with the same name), (c) states and salary brackets determine tax rates (a tax rate depends on both the state and employee salary), etc. We varied our CFDs using the following parameters: NUMCFDs determined the number of CFDs considered in an experimental setup, NUMATTRs the (max) attribute number in the CFDs, TABSZ the (max) tuple number in the CFDs, and NUMCONSTs the percentage of tuples with constants vs. tuples with variables in each CFD.

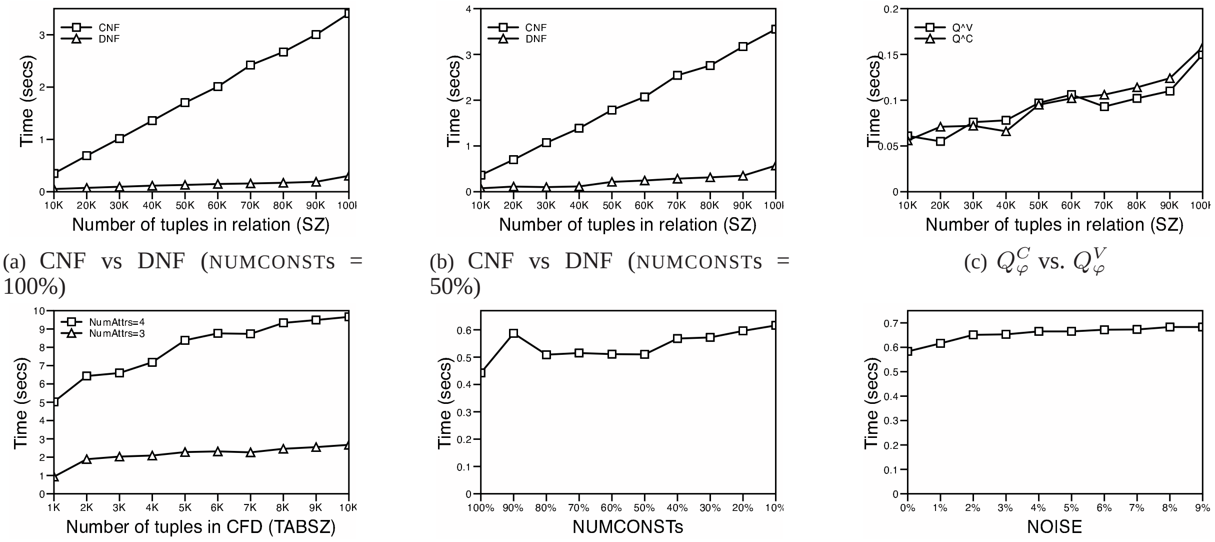
SQL query evaluation: There are two alternative evaluation strategies for the SQL detection queries of Section 4. Key distinction between these two strategies is how we evaluate the **where** clause in each detection query. Specifically, note that the **where** clause of our SQL detection queries is in conjunctive normal form (CNF). It is known that database systems do not execute efficiently queries in CNF since the presence of the OR operator leads the optimizer to select inefficient plans that do not leverage the available indexes. A solution to this problem is to convert conditions in the **where** clause into disjunctive normal form (DNF). This conversion might cause an exponential blow-up in the number of conjuncts, but in this case, the blow-up is *w.r.t.* the number of attributes in the CFD, which is usually very small.

CNF **vs.** DNF**:** In this experiment We considered both evaluation strategies, under various settings, to determine the most efficient one. In more detail, we considered relations with SZ from 10K to 100K tuples, in 10K increments, and 5% NOISE. We considered two *representative* CFDs, each with NUMATTRs 3, where the first CFD had NUMCONSTs 100% (tuples with only constant) while the second had NUMCONSTs 50% (half the tuples had variables). In terms of CFD size, we set TABSZ to 1K. Figures 9(a) and 9(b) show the evaluation times for both evaluation strategies, for each of the two CFDs. As both graphs show, irrespective of data size and the presence of constants or variables, the DNF strategy clearly out-performs the CNF one. Furthermore, the figures illustrate the scalability of our detection queries SZ.

Q^C_φ **vs.** Q^V_φ**:** In this experiment, we investigated how the detection time is split between the Q^C_φ and Q^V_φ queries. We considered relations with SZ from 10K to 100K tuples, in 10K increments, and 5% NOISE. For the CFD, we consider one with NUMATTRs equal to 3, TABSZ to 1K and NUMCONSTs 100% (we made similar observations for other values of NUMCONSTs). Figure 9(c) shows the evaluation times for each query in isolation and shows that both queries have similar loads and they follow the same execution trend.

Scalability in TABSZ**:** This was to study the scalability of the detection queries with respect to TABSZ. In more detail, we fixed SZ to 500K with 5% NOISE. We considered two CFDs whose sizes varied from 1K to 10K, in 1K increments. The NUMATTRs was 3 for the first, and 4 for the second CFD considered. For all CFDs, NUMCONSTs was 50%. Figure9(d) shows the detection times for the 2 CFDs. As is obvious from the figure, TABSZ has little impact on the detection times and dominant factors here are (a) the size of the relation, which is much larger than the tableaux, and (b) the number of attributes in the tableau, since these result in more complicated join conditions in the detection queries.

Scalability in NUMCONSTs**:** We studied the impact of variables on the detection times. We considered a relation with SZ 100K and NOISE 5% and a CFDs with TABSZ 1K, and NUMATTRs = 3. We varied NUMCONSTs between 100% (all constants) and 10% and we measured the detection times over the relation. Figure 9(e) shows that variables do affect detection times and (not shown in the figure) moreover, as



(a) CNF vs DNF (NUMCONSTs = 100%)

(b) CNF vs DNF (NUMCONSTs = 50%)

(c) Q^C_φ vs. Q^V_φ

(d) Scalability in TABSZ

(e) Scalability in NUMCONSTs

(f) Scalability in NOISE

Figure 9. Experimental results

we increased both the percentage of variables and the number of attributes with variables, detection times increased noticeably. This is apparent, given that variables restrict the use of indexes while joining the relation with the tableau.

Scalability in NOISE: Here, we varied NOISE between 0% and 9% in a relation with SZ 100K, and we measured detection time, for a CFD with TABSZ 30K (we used all possible zip to state pairs, so as not to miss a violation), NUMATTRs 2, and NUMCONSTs 100%. As we can see in Fig.9(f), the level of NOISE has negligable effects on detection times.

Merging CFDs: Our experiments indicate (not shown) that CFD merging is mainly beneficial for highly-related CFDs, as might be expected. However, the performance of the merged scheme is hampered by the difficulty faced by our optimizers when handling **where** clauses in CNF. The conversion to DNF is not an option here, because each disjunct in CNF consists of 3 terms, and thus the translation of CNF to DNF results in a **where** clause with 3^k conjuncts, where k is the number of attributes in the CFD. In practice this is much worse than the 2^k increase that results from translating Q^C_φ or Q^V_φ into DNF. We speculate that improvements in CNF evaluation may make the merge technique more useful. Also, we are investigating techniques to factor out work from multiple CFD violation detections in a manner similar to [4], and expect the merged representation to be a convenient basis for optimizations of this form.

6 CFD Repairing: Discussion

To clean data, an effective method should be in place for removing inconsistencies from the data, in addition to inconsistency detection. That is, if a database I of a relation schema R violates a set Σ of CFDs on R, we want to find a *minimal repair* I' of I such that $I' \models \Sigma$ and I' minimally differs from the original database I, obtained by performing repair operations on I [2]. Following [3, 7, 20] we allow attribute-value modifications as repair operations.

Repairing CFDs is nontrivial. Indeed, consider the CFD *repairing problem*, which is to determine, given I, Σ and a positive integer k, whether or not there exists an instance I' of R such that (1) $I' \models \Sigma$ and (2) I' is obtained from I by at most k repair operations. This problem is intractable (by reduction from the set cover problem; see [8] for a proof).

Theorem 6.1: *The repairing problem is* NP*-complete.* □

Repairing CFDs introduces challenges not encountered when repairing standard FDs. For example, it is known [3] that if a pair t_1, t_2 of tuples violate an FD $X \rightarrow A$, one can resolve the inconsistency by modifying $t_1[A]$ or $t_2[A]$ such that $t_1[A] = t_2[A]$. That is, FD violations can always be resolved by modifying values of some t_1 (or t_2) attributes in the RHS of FDs, without changing the LHS such that $t_1[X] \neq t_2[X]$. In contrast, this strategy no longer works for CFDs. To see this, consider a schema R with $\mathtt{attr}(R) = (A, B, C)$, an instance I of R consisting of (a_1, b_1, c_1) and (a_1, b_2, c_2), and a set Σ of CFDs including $(A \rightarrow B, (\_, \_))$ and $(C \rightarrow B, \{(c_1, b_1), (c_2, b_2)\})$. Then $I \not\models \Sigma$ and moreover, any repair I' has to modify values of some attributes in the LHS of the FDs embedded in the CFDs.

In light of Theorem 6.1 we have developed a heuristic algorithm for finding a repair of a database, overcoming the new challenges. We defer report on the heuristic pending the completion of implementation and experimental study.

7 Concluding Remarks

We have introduced CFDs and shown that CFDs can express semantics of data fundamental to data cleaning. For reasoning about CFDs we have provided techniques and a sound and complete inference system for their consistency and implication analyses. For applications of CFDs in data cleaning, we have developed SQL-based techniques for detecting inconsistencies as violations of CFDs. We have also experimentally evaluated our detection techniques.

There has been work on data cleaning based on constraints (e.g., [2, 5, 7, 20]). Research in this area has mostly focused on two topics, both introduced in [2]: *repair* is to find another database that is consistent and minimally differs from the original database (*e.g.*, [2, 5, 7]); and *consistent query answer* is to find an answer to a given query in every repair of the original database (*e.g.*, [2, 20]). Most earlier work (except [7, 20]) considers traditional full (subsuming functional) dependencies and denial constraints, which do not allow patterns with data values and are quite different from CFDs. Beyond traditional dependencies, logic programming is studied in [7] for fixing census data. Closer to CFDs is the tableau representation of full dependencies, which also allow data values [20]. The work of [20] differs from ours in that it focuses on condensed representation of repairs and consistent query answers. As remarked in Section 2, a class of instance-level FDs is studied in [14], which are a special case of CFDs. [14, 7, 20] consider neither consistency analysis and inference system, nor detection of inconsistencies by means of SQL queries.

Codd tables, variable tables and conditional tables have been traditionally used in the context of incomplete information [12, 10]. The key difference between these table formalisms and pattern tableaux in CFDs is that each of these tables is used as a representation of possibly infinitely many relation instances, one instance for each instantiation of variables. No instance represented by these table formalisms can include two tuples that result from different instantiations of a table tuple. In contrast, a pattern tableau is used to constrain–as part of a CFD–a *single* relation instance, which can contain any number of tuples that are all instantiations of the same pattern tuple. Closer to our pattern tableau is the notion of *mapping tables* studied for data sharing [13], for which no inference system is developed.

As remarked in Section 1, there have been ETL tools (see [17] for a comprehensive survey) and merge-purge algorithms (*e.g.*, [9, 11, 16, 21]) for data cleaning. Related to our work are also the AJAX system [9] which proposes a declarative language for specifying data cleaning programs, and the Potter's Wheel system [18] that extracts structure for attribute values and uses these to flag discrepancies in the data. While a constraint repair facility will logically become part of the cleaning process supported by these tools and systems, we are not aware of analogous functionality currently in any of the systems mentioned.

There is naturally much more to be done. First, as remarked in Section 6 we are implementing our CFD-based algorithms for repairing CFDs. Second, to clean data, constraints beyond CFDs are certainly needed. We are studying data cleaning based on both CFDs and conditional inclusion dependencies. Third, we are developing automated methods for discovering CFDs and repairing inconsistent CFDs.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, 1999.

[3] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, 2005.

[4] Z. Chen and V. Narasayya. Efficient computation of multiple group by queries. In *SIGMOD*, 2005.

[5] J. Chomicki and J. Marcinkowski. "Minimal-Change Integrity Maintenance Using Tuple Deletions". *Information and Computation*, 197(1-2):90–121, 2004.

[6] W. W. Eckerson. Data Quality and the Bottom Line: Achieving Business Success through a Commitment to High Quality Data. Technical report, The Data Warehousing Institute, 2002. *http://www.tdwi.org/research/display.aspx?ID=6064*.

[7] E. Franconi, A. L. Palma, N. Leone, S. Perri, and F. Scarcello. Census data repair: a challenging application of disjunctive logic programming. In *LPAR*, 2001.

[8] Full version. *http://www.lfcs.inf. ed.ac.uk/research/ database/publications/CFD.pdf*.

[9] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C. Saita. "AJAX: An Extensible Data Cleaning Tool". In *SIGMOD*, 2001.

[10] G. Grahne. *The Problem of Incomplete Information in Relational Databases*. Springer, 1991.

[11] M. A. Hernandez and S. Stolfo. "Real-World Data is Dirty: Data Cleansing and the Merge/Purge Problem". *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[12] T. Imieliński and W. L. Jr. Incomplete information in relational databases. *JACM*, 31(4):761–791, 1984.

[13] A. Kementsietsidis, M. Arenas, and R. J. Miller. Data mapping in P2P systems: Semantics and algorithmic issues. In *SIGMOD*, 2003.

[14] E. Lim and S. Prabhakar. Entity identification in database integration. In *ICDE*, 1993.

[15] D. Maier. Minimum covers in relational database model. *J. ACM*, 27(4):664–674, 1980.

[16] A. Monge. Matching algorithm within a duplicate detection system. *IEEE Data Engineering Bulletin*, 23(4), 2000.

[17] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4), 2000.

[18] V. Raman and J. M. Hellerstein. "Potter's Wheel: An Interactive Data Cleaning System". In *VLDB*, 2001.

[19] C. C. Shilakes and J. Tylman. Enterprise information portals, Nov. 1998.

[20] J. Wijsen. Condensed representation of database repairs for consistent query answering. In *ICDT*, 2003.

[21] W. Winkler. Advanced methods for record linkage. Technical report, Statistical Research Division, U.S. Bureau of the Census., 1994.