Discovering Data Quality Rules [*]

Fei Chiang
University of Toronto
fchiang@cs.toronto.edu

Renée J. Miller
University of Toronto
miller@cs.toronto.edu

ABSTRACT

Dirty data is a serious problem for businesses leading to incorrect decision making, inefficient daily operations, and ultimately wasting both time and money. Dirty data often arises when domain constraints and business rules, meant to preserve data consistency and accuracy, are enforced incompletely or not at all in application code.

In this work, we propose a new data-driven tool that can be used within an organization's data quality management process to suggest possible rules, and to identify conformant and non-conformant records. Data quality rules are known to be contextual, so we focus on the discovery of context-dependent rules. Specifically, we search for conditional functional dependencies (CFDs), that is, functional dependencies that hold only over a portion of the data. The output of our tool is a set of functional dependencies together with the context in which they hold (for example, a rule that states for CS graduate courses, the course number and term functionally determines the room and instructor). Since the input to our tool will likely be a dirty database, we also search for CFDs that almost hold. We return these rules together with the non-conformant records (as these are potentially dirty records).

We present effective algorithms for discovering CFDs and dirty values in a data instance. Our discovery algorithm searches for minimal CFDs among the data values and prunes redundant candidates. No universal objective measures of data quality or data quality rules are known. Hence, to avoid returning an unnecessarily large number of CFDs and only those that are most interesting, we evaluate a set of interest metrics and present comparative results using real datasets. We also present an experimental study showing the scalability of our techniques.

1. INTRODUCTION

Poor data quality continues to be a mainstream issue for many organizations. Having erroneous, duplicate or incomplete data leads to ineffective marketing, operational inefficiencies, inferior customer relationship management, and poor business decisions. It is estimated that dirty data costs US businesses over $600 billion a year [11]. There is an increased need for effective methods to improve data quality and to restore consistency.

Dirty data often arises due to changes in use and perception of the data, and violations of integrity constraints (or lack of such constraints). Integrity constraints, meant to preserve data consistency and accuracy, are defined according to domain specific business rules. These rules define relationships among a restricted set of attribute values that are expected to be true under a given context. For example, an organization may have rules such as: (1) all new customers will receive a 15% discount on their first purchase and preferred customers receive a 25% discount on all purchases; and (2) for US customer addresses, the street, city and state functionally determines the zipcode. Deriving a complete set of integrity constraints that accurately reflects an organization's policies and domain semantics is a primary task towards improving data quality.

To address this task, many organizations employ consultants to develop a data quality management process. This process involves looking at the current data instance and identifying existing integrity constraints, dirty records, and developing new constraints. These new constraints are normally developed in consultation with users who have specific knowledge of business policies that must be enforced. This effort can take a considerable amount of time. Furthermore, there may exist domain specific rules in the data that users are not aware of, but that can be useful towards enforcing semantic data consistency. When such rules are not explicitly enforced, the data may become inconsistent.

Identifying inconsistent values is a fundamental step in the data cleaning process. Records may contain inconsistent values that are clearly erroneous or may potentially be dirty. Values that are clearly incorrect are normally easy to identify (e.g., a 'husband' who is a 'female'). Data values that are potentially incorrect are not as easy to disambiguate (e.g., a 'child' whose yearly 'salary' is '$100K'). The unlikely co-occurrence of these values causes them to become dirty candidates. Further semantic and domain knowledge may be required to determine the correct values.

For example, Table 1 shows a sample of records from a 1994 US Adult Census database [4] that contains records of citizens and their workclass (CLS), education level (ED), marital status (MR), occupation (OCC), family relationship (REL), gender (GEN), and whether their salary (SAL) is

[*]Supported in part by NSERC and NSF 0438866.

Permission to make digital or hard copies of portions of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright for components of this work owned by others than VLDB Endowment must be honored.
Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists requires prior specific permission and/or a fee. Request permission to republish from: Publications Dept., ACM, Inc. Fax +1 (212) 869-0481 or permissions@acm.org.

PVLDB '08, August 23-28, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 978-1-60558-305-1/08/08

Table 1: Records from US 1994 Adult Census database

Tuple	CLS	ED	MR	OCC	REL	GEN	SAL
t_1	Private	Bachelors	Married	Exec-mgr	Husband	Male	>50K
t_2	Private	Bachelors	Married	Prof-specialty	Husband	Male	>50K
t_3	Self-emp	Masters	Married	Exec-mgr	Wife	Male	>50K
t_4	?	HS-grad	Divorced	?	Not-in-family	Male	>50K
t_5	Self-emp	Masters	Married	Admin	Wife	Female	>50K
t_6	Never-worked	7th-8th	Divorced	?	Not-in-family	Male	≤50K
t_7	Self-emp	HS-grad	Never-married	Farming	Own-child	Male	≤50K
t_8	Local-gov	Some-college	Never-married	Admin	Own-child	Female	≤50K
t_9	State-gov	Masters	Divorced	Prof-specialty	Own-child	Male	>50K
t_{10}	?	Bachelors	Divorced	?	Not-in-family	Female	≤50K
t_{11}	Self-emp	Some-college	Never-married	Machine-op	Not-in-family	Female	>50K

above or below \$50K. We intuitively recognize the following inconsistencies:

1. In t_3, if the person is a married male, then REL should be $Husband$ not $Wife$. Alternatively, the GEN should be $Female$. This is clearly an erroneous value.
2. The tuple t_6 indicates a person with a missing occupation (where the "?" indicates a NULL) and a workclass indicating he has never worked. Support in the data reveals that when occupation is missing so is the workclass value, indicating that if the person is temporarily unemployed or their occupation is unknown, so is their workclass. Given this evidence, the discrepancy in t_6 indicates either the value "Never-worked" is dirty and should be "?"; or a semantic rule exists to distinguish people who have never been employed and their salary must be less than \$50K (versus those who are temporarily unemployed or their occupation is unknown).
3. The tuple t_9 represents a child (under 18) who is divorced with a salary over \$50K. Most children are unmarried and do not have such a high salary.

Cases (2) and (3) are potential exceptions whereby an analyst with domain knowledge can help to resolve these inconsistencies, and may enforce new rules based on these discrepancies.

In this paper, we propose a new data-driven tool that can be used during the data quality management process to suggest possible rules that hold over the data and to identify dirty and inconsistent data values. Our approach simplifies and accelerates the cleaning process, facilitating an interactive process with a consultant who can validate the suggested rules (including rules that may not be obvious to users) against business requirements so that they are enforced in the data, and with a data analyst who can resolve potential inconsistencies and clean the obvious dirty values. As described above, data quality rules are known to be contextual, so we focus on the discovery of context-dependent rules. In particular, we search for *conditional functional dependencies* (CFDs) [5], which are functional dependencies that hold on a subset of the relation (under a specific context). For example, the earlier business rules can be expressed as CFDs:

- ϕ_{1a} : `[status = 'NEW', numPurchases = 1]` $\rightarrow$ `[discount = 15%]`
- ϕ_{1b} : `[status = 'PREF']` $\rightarrow$ `[discount = 25%]`
- ϕ_2 : `[CTRY = 'US', STR, CTY, ST]` $\rightarrow$ `[ZIP]`

Similarly, the semantic quality rules for the Census data can be represented as:

- ϕ_3 : `[MR = 'Married', GEN = 'Male']` $\rightarrow$ `[REL = 'Husband']`
- ϕ_4 : `[CLS = 'Never-worked']` $\rightarrow$ `[OCC = '?', SAL = '`$\leq$` $50K']`
- ϕ_5 : `[REL = 'Own-child']` $\rightarrow$ `[MR = 'Never-married', SAL = '`$\leq$` $50K']`

In this paper we formalize and provide solutions for discovering CFDs and for identifying dirty data records. In particular, we make the following contributions:

- We present an algorithm that effectively discovers CFDs that hold over a given relation R. We prune redundant candidates as early as possible to reduce the search space and return a set of minimal CFDs.
- We formalize the definition of an exception (dirty) value and present an algorithm for finding exceptions that violate a candidate CFD. Exceptions are identified efficiently during the discovery process. Once found, they can be reported back to an analyst for verification and correction.
- To avoid returning an unnecessarily large number of CFDs and only ones that are most interesting, we evaluate a set of interest metrics for discovering CFDs and identifying exceptions. Our results show that conviction is an effective measure for identifying intuitive and interesting contextual rules and dirty data values.
- We present an experimental evaluation of our techniques demonstrating their usefulness for identifying meaningful rules. We perform a comparative study against two other algorithms showing the relevancy of our rules, and evaluate the scalability of our techniques for larger data sizes.

The rest of the paper is organized as follows. In Section 2, we present related work and in Section 3 we give preliminary definitions and details of our rule discovery algorithm. In

Section 4, we present our algorithm for identifying records that are exceptions to approximate rules and lead to dirty data values. Section 5 presents an experimental evaluation of our methods and Section 6 concludes our paper.

2. RELATED WORK

Our work finds similarities to three main lines of work: functional dependency discovery, conditional functional dependencies, and association rule mining.

Functional dependency (FD) discovery involves mining for all dependencies that hold over a relation. This includes discovery of functional [13, 18, 24], multi-valued [22] and approximate [13, 16] functional dependencies. In previous FD discovery algorithms, both Tane [13] and DepMiner [18] search the attribute lattice in a levelwise manner for a minimal FD cover. FastFDs [24] uses a greedy, heuristic, depth-first search that may lead to non-minimal FDs, requiring further checks for minimality. We generalize the lattice based levelwise search strategy for discovering CFDs.

Conditional functional dependencies, are a form of constrained functional dependencies (introduced by Maher [19]). Both Maher and Bohannon et al. [5] extend Armstrong's axioms to present a (minimal) set of inference rules for CFDs. To facilitate data cleaning, Bohannon et al. [5] provide SQL based techniques for detecting tuples that violate a given set of CFDs. Both of these papers have assumed that the CFDs are known, which is not always true. Manual discovery of CFDs is a tedious process that involves searching the large space of attribute values. None of the previous work has focused on this problem.

Recent work in conditional dependencies has focused on conditional inclusion dependencies (CIND) and finding repairs for tuples violating a given set of CFDs. Inclusion dependencies are extended with conditional data values to specify that the inclusion dependencies hold over a subset of the relation. Bravo et al. [6] study the implication and consistency problems associated with CINDs and their interaction with CFDs, providing complexity results, inference rules and a heuristic consistency-check algorithm. Constraint based data cleaning searches for a repair database D' that is consistent with the CFDs and differs minimally from the original database D. Cong et al.[9] propose repair algorithms to repair D based on the CFDs and potential updates to D. We focus on discovering CFDs and identifying tuples that violate these rules. Any of the proposed repair algorithms can be used to clean the reported dirty tuples.

There are many statistical approaches for error localization, again, assuming the data quality rules are known (and valid) where the goal is to find errors or rule violations [23]. Many data cleaning tools support the discovery of statistical trends and potential join paths [10], schema restructuring and object identification [21, 12], but to the best of our knowledge none of these tools focus on the discovery of contextual dependency rules.

Association rule mining (ARM) focuses on identifying relationships among a set of items in a database. The semantics of association rules differ from CFDs. In ARM, for itemsets U, V, the rule $U \to V$ indicates U occurs with V. However, U may also occur with other elements. CFDs define a strict semantics where for a set of attributes X, Y, a CFD $X \to Y$ indicates that the values in X must (only) occur with the corresponding values in Y. Large itemsets that satisfy minimum support and confidence thresholds are most interesting [1, 2]. Statistical significance and conviction tests [7] have also been used to assess itemset quality. We employ some of these measures to evaluate the quality of a CFD during our discovery process.

Table 2: CFD ϕ: ([`STAT, NUM-PUR`] $\to$ [`DISC`], T_1)

STAT	NUM-PUR	DISC
'New'	1	15
'Pref'	'−'	25

One of the foundational techniques in ARM, the Apriori algorithm [3], builds large itemsets by adding items only to large itemsets and pruning small itemsets early that do not satisfy the support threshold. A superset of a small itemset will remain small, hence the itemset can be pruned to reduce the number of itemsets considered. This is known as the *anti-monotonic property.* In our work, applying this pruning technique to our classes of data values may lead us to miss groupings of smaller constant values that together may form a valid condition satisfying the support threshold. We eliminate conditioning on attributes whose maximum class size is less than the support level since their supersets will also fall below the support level.

Related to ARM, frequent pattern mining searches for frequent itemsets that satisfy domain, class, or aggregate constraints [14, 15, 20, 17]. This helps to focus the mining on desired patterns. A primary objective is to discover data patterns involving *similar* values defined by (given) domain widths [14, 15]. Similar to the anti-monotonic property, optimizations for computing frequent patterns involve constraint push down [20], and pruning strategies that are specific to the constraint type [17]. Both ARM and frequent pattern mining focus on value based associations, whereas our techniques focus on finding rules that allow variables not just constant values.

3. SEARCHING FOR DATA QUALITY RULES

Given a relation R, we are interested in finding all data quality rules that exist in R and that satisfy some minimum threshold. As data quality rules are contextual, we search for conditional functional dependencies (CFD), which are functional dependencies that hold under certain conditions. We first give some preliminary definitions.

3.1 Preliminaries

A CFD ϕ over R can be represented by $\phi : (X \to Y, T_p)$ [5], where X, Y are attribute sets in R and $X \to Y$ is a functional dependency (FD). T_p is a pattern tableau of ϕ, containing all attributes in X and Y. For each attribute B in $(X \cup Y)$, the value of B for a tuple in T_p, $t_p[B]$, is either a value from the domain of B or '−', representing a variable value. For example, the rules ϕ_{1a} and ϕ_{1b} are represented as rows in Table 2. A CFD enhances functional dependencies by allowing constant values to be associated with the attributes in the dependency. In particular, CFDs are able to capture semantic relationships among the data values.

A tuple t *matches* a tuple t_p in tableau T_p if for each attribute B in T_p, for some constant 'b', $t[B] = t_p[B] =$ 'b' or $t_p[B] =$ '−'. For example, the tuple (`'Pref',10,25`) matches the second tuple in T_1 (in Table 2). A subset I of a relation R satisfies the CFD ϕ, denoted as $I \models \phi$, if for every pair of tuples t_1 and t_2 in I, and for each pattern tuple t_p in T_p, if $t_1[B] = t_2[B]$ for every attribute B in X, and both t_1, t_2 match $t_p[B]$, then $t_1[A] = t_2[A]$ and t_1, t_2 both match $t_p[A]$ for attribute A in Y. Tuple t in R violates ϕ if $t[B] = t_p[B] =$ 'b' or $t_p[B] =$ '$-$' for every attribute B in X, but $t[A] \neq t_p[A]$ for attribute A in Y. Without loss of generality, we consider CFDs of the form $\varphi : (X \rightarrow A, t_p)$, where A is a single attribute and t_p is a single pattern tuple. The general form $\phi : (X \rightarrow Y, T_p)$ can be represented in this simpler form as $(X \rightarrow A, t_p(X \cup A))$ for each A in Y and t_p in T_p. Intuitively, we decompose Y into its singular attributes and consider each resulting dependency individually. This is equivalent to applying the decomposition rule[1] [5, 19]. CFDs of this form are *minimal* (in standard form) when A does not functionally depend on any proper subset of X so that each attribute in X is necessary for the dependency to hold. This structural definition also holds for FDs.

We search for CFDs where X may contain multiple variables P and $(X - P)$ conditional attributes. For $X \rightarrow A$, if the consequent A resolves to a constant 'a' for all values in variable(s) P ($t_p[P] =$ '$-$'), then we report 'a' in t_p and remove P from the rule since it has no effect on the value of A. Returning a constant value rather than a variable A, along with the conditional values, enables the rules to provide increased information content.

In the next section, we describe the attribute partition model and how quality rules are generated. We then present the metrics that we use to evaluate the quality of a discovered rule, followed by a discussion of the pruning strategies we adopt to narrow the search space and to avoid redundant candidates.

3.2 Generating Candidates

Let n be the number of attributes in R and let X be an attribute set of R. A *partition* of X, Π_X, is a set of equivalence classes where each class contains all tuples that share the same value in X. Let x_i represent an equivalence class with a representative i that is equal to the smallest tuple id in the class. Let $|x_i|$ be the size of the class and v_i be the distinct value of X that the class represents. For example, in Table 1 for $X_1 = \texttt{MR}$, $\Pi_{\texttt{MR}} = \{\{1,2,3,5\},\{4,6,9,10\},\{7,8,11\}\}$, for X_2=REL, $\Pi_{\texttt{REL}} = \{\{1,2\},\{3,5\},\{4,6,10,11\},\{7,8,9\}\}$, and for $X_3 = (\texttt{MR,REL})$, $\Pi_{(\texttt{MR, REL})} = \{\{1,2\},\{3,5\},\{4,6,10\},\{9\},\{11\},\{7,8\}\}$. For Π_{MR}, $x_7 = \{7,8,11\}$, $|x_7| = 3$, and $v_7 =$ `'Never-married'`. A partition Π_X *refines* Π_Y, if every x_i in Π_X is a subset of some y_i in Π_Y [13]. The FD test is based on partition refinement where a candidate FD $X \rightarrow Y$ over R holds iff Π_X refines Π_Y. Since CFDs hold for only a portion of R, the CFD test does not require complete refinement, that is, it does not require every class in Π_X to be a subset of some class in Π_Y. We will elaborate on this shortly.

To find all minimal CFDs, we search through the space of non-trivial[2] and non-redundant candidates, where each candidate is of the form $\gamma : X \rightarrow Y$, and test whether γ holds under a specific condition. A candidate γ is non-redundant if its attribute sets X, Y are not supersets of already discovered CFDs. The set of possible antecedent values is the collection of all attribute sets, which can be modeled as a set containment lattice, as shown in Figure 1. Each node in the lattice represents an attribute set and an edge exists between sets X and Y if $X \subset Y$ and Y has exactly one more attribute than X, i.e., $Y = X \cup \{A\}$. We define a candidate CFD based on an edge (X, Y) as follows.

[1]Decomposition rule: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
[2]A candidate $X \rightarrow A$ is trivial if $A \subseteq X$.

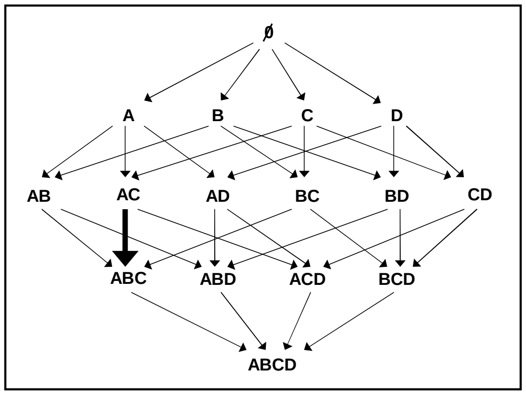

Figure 1: Attribute search lattice

Definition 1. *An edge (X, Y) in the lattice generates a candidate CFD $\gamma : ([Q, P] \rightarrow A)$ consisting of variable attributes P and $Q = (X - P)$ conditional attributes, where $Y = (X \cup A)$. P and Q consist of attribute sets that range over the parent nodes of X in the lattice.*

For example, in Figure 1, for edge $(X, Y) = (\texttt{AC, ABC})$, we consider conditional rules γ_1: $([Q = \texttt{A}, \texttt{C}] \rightarrow \texttt{B})$ and γ_2: $([Q = \texttt{C}, \texttt{A}] \rightarrow \texttt{B})$, where Q ranges over the parent nodes of AC, and will take on values from these attribute sets. For the single attribute case, $(X, Y) = (\texttt{A, AB})$, we consider the conditional rule γ: $([Q = \texttt{A}, \emptyset] \rightarrow \texttt{B})$. For the rest of the paper, we will use this notation (based on attribute sets) to represent candidate CFDs.

Each candidate γ is evaluated by traversing the lattice in a breadth-first search (BFS) manner. We first consider all X consisting of single attribute sets (at level $k = 1$), followed by all 2-attribute sets, and we continue level by level to multi-attribute sets until (potentially) level $k = n - 1$. We only visit nodes on the BFS stack that represent minimal CFD candidates. The algorithm efficiency is based on reducing the amount of work in each level by using results from previous levels. In particular, by pruning nodes that are supersets of already discovered rules and nodes that do not satisfy given thresholds, we can reduce the search space (by avoiding the evaluation of descendant nodes) thereby saving considerable computation time.

CFD validity test For attribute sets X, Y a class x_i in Π_X is *subsumed* by y_i in Π_Y if x_i is a subset of y_i. Let $\Omega_X \subseteq \Pi_X$ represent all the subsumed classes in Π_X. Let $X = (P \cup Q)$ where P represents variable attributes and Q the conditional attributes, then $\gamma : ([Q, P] \rightarrow A)$ is a CFD iff there exists a class q_i in Π_Q that contains values *only* from Ω_X.

Intuitively, the CFD validity test is based on identifying values in X that map to the same Y value. Since values are modeled as equivalence classes, we are interested in x_i that do not split into two or more classes in Y (due to attribute A). If there is a condition Q that the tuples in Ω_X share in common, then γ is true under Q. For data quality rules, it is generally preferable to find rules that hold for more than one, or indeed for many, tuples. Hence, we may additionally impose a constraint that $|x_i| > l$ for some threshold value l.

Example 1. *The search for CFDs in Table 1 begins at level $k = 1$ and we set $l = 1$. Consider the following partitions (for brevity, we show only a subset of the attribute partitions):*

- $\Pi_{MR} = \{\{1,2,3,5\},\{4,6,9,10\},\{7,8,11\}\}$
- $\Pi_{REL} = \{\{1,2\},\{3,5\},\{4,6,10,11\},\{7,8,9\}\}$
- $\Pi_{(R,M)}{}^{3} = \{\{1,2\},\{3,5\},\{4,6,10\},\{7,8\},\{9\},\{11\}\}$
- $\Pi_{ED} = \{\{1,2,10\},\{3,5,9\},\{4,7\},\{6\},\{8,11\}\}$
- $\Pi_{CLS} = \{\{1,2\},\{3,5,7,11\},\{4,10\},\{6\},\{8\},\{9\}\}$
- $\Pi_{(C,E)} = \{\{1,2\},\{10\},\{3,5\},\{4\},\{6\},\{7\},\{8\},\{9\},\{11\}\}$
- $\Pi_{(M,E)} = \{\{1,2\},\{10\},\{3,5\},\{9\},\{4\},\{6\},\{7\},\{8,11\}\}$
- $\Pi_{(M,E,C)} = \{\{1,2\},\{10\},\{3,5\},\{9\},\{4\},\{6\},\{7\},\{8\},\{11\}\}$

For edge (X,Y) = (REL, (REL, MR)), x_1 and x_3 are the subsumed classes. Since X is a single attribute, if we take $Q = X$ = REL and $P = \emptyset$, then we can condition Q on v_1, v_3, to obtain the rules φ_1: ([REL = 'Husband'] $\rightarrow$ MR) and φ_2: ([REL = 'Wife'] $\rightarrow$ MR) as CFDs. Since the values in x_1 and x_3 each map to the constant 'Married', we return the more informative rule φ_1 : ([REL = 'Husband'] $\rightarrow$ [MR = 'Married']) (similarly for φ_2), indicating that if a person is a husband, then he must be married.

Consider the edge (X,Y) = (ED, (ED, CLS)), there are no subsumed x_i classes. After evaluating edges from level-1, we consider edges from level-2 nodes. Now consider the edge (X,Y) = ((ED,MR), (ED,MR,CLS)). The classes x_1 and x_3 are subsumed and contained in $\Omega_{(ED,MR)}$. We check if there exists a class q_i in $\Pi_{Q=MR}$ that contains only values from x_1 and x_3. We see that q_1 ={1,2,3,5} satisfies this requirement. Therefore, by the CFD test we return φ: ([MR = 'Married', ED] $\rightarrow$ [CLS]) as a conditional data quality rule.

The traversal continues to nodes representing CFD candidates that require further evaluation. We use information about discovered CFDs from previous levels to prune the search space and avoid evaluating redundant candidates. We will elaborate on this pruning strategy in Section 3.5.

3.3 Multiple Variables and Conditional Attributes

The CFD φ is true if there exists some value q (condition on Q) that holds over Ω_X. The remaining non-subsumed classes $\Lambda_X = (\Pi_X - \Omega_X)$ are not considered during the conditioning. If there is no conditional value q that is true over Ω_X, then there does not exist a dependency with conditioning attributes Q. However, the classes in Ω_X can still be refined by considering additional variable and conditional attributes. In particular, we consider the updated set of attributes X' (with added variables and conditions), and test whether a rule exists over the classes in $\Omega_{X'}$.

First, we consider adding another variable attribute B. Considering an extra attribute can help to further restrict

[3]For brevity, we abbreviate (REL,MR) to (R,M)

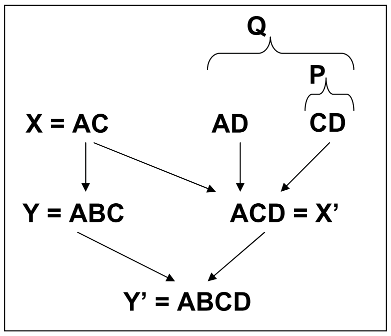

Figure 2: A portion of the attribute search lattice

classes that previously were not subsumed. When a candidate $\gamma : X \rightarrow A$ corresponding to an edge (X, Y) fails to materialize to a CFD, we generate a new candidate $X' \rightarrow Y'$ in the next level of the lattice where $X' = (X \cup B)$ and $Y' = (Y \cup B)$, $B \neq A$. The variable attributes P range over the attribute sets that are parents of X' and $(X' - P)$ is the current conditional attribute. See Figure 2 for an example. If γ: ([A,C] $\rightarrow$ B) with conditional node A, corresponding to edge (X,Y) = (AC, ABC) does not materialize to a CFD, a new candidate based on the edge (X',Y') will be evaluated. Specifically, γ': ([A,C,D] $\rightarrow$ B) will be considered with variable attributes C,D.

If a candidate rule does not materialize to a CFD after adding a variable attribute, then we condition on an additional attribute. Similar to adding variables, we consider conditional attributes Q that range over the attribute sets that are parents of X' (see Figure 2). We add these new candidates (X',Y') to a global candidate list G and mark the corresponding lattice edge (X',Y'). We perform the CFD test only for classes in $\Omega_{X'}$. At level $k \geq 2$, where multi-condition and multi-variable candidates are first considered, we only visit nodes with marked edges to ensure minimal rules are returned, and thereby also reducing the search space.

Example 2. *Continuing from Example 1, the evaluation of edge (X,Y) = ((REL), (REL, MR)) produced classes x_4 and x_7 that were not subsumed. Consider edge (X',Y') = (SAL,REL), (SAL, REL, MR)), with γ: [Q = (SAL), REL] $\rightarrow$ [MR]. The attribute partitions are:*

- $\Pi_{SAL} = \{\{1,2,3,4,5,9,11\},\{6,7,8,10\}\}$
- $\Pi_{(SAL,REL)} = \{\{1,2\},\{3,5\},\{4,11\},\{9\},\{6,10\},\{7,8\}\}$
- $\Pi_{(S,R,M)} = \{\{1,2\},\{3,5\},\{4\},\{11\},\{9\},\{6,10\},\{7,8\}\}$

Classes x_1, x_3, x_6, x_7 are subsumed classes. However, we do not condition using x_1 and x_3 since a CFD was found previously with these classes. We consider only x_6 and x_7 in $\Omega_{(X=SAL,REL)}$. Under Q = (SAL), we check for a q_i in Π_Q that contains only tuple values from Ω_X. The class q_1 fails since it contains tuple values not in Ω_X, but q_6 = {6,7,8,10} with v_6 = '≤50K' qualifies. All tuple values in x_6 and x_7 share the same conditional value v_6, hence, [SAL = '≤50K', REL] $\rightarrow$ [MR] is a conditional data quality rule.

After visiting all marked edges, the set of discovered rules and their conditional contexts are reported to the user. Algorithm 1 presents pseudocode to identify conditional data quality rules over R.

Algorithm 1 Algorithm to discover CFDs

```
 INPUT Relation $R$, current level $k$
 1: Initialize $CL = \{\}$, $G = \{\}$
 2: for each $X$ in level $k$ do
 3:    consider (marked) edge $(X, Y)$
 4:    if $|\Pi_X| = |\Pi_Y|$ then
 5:       $\{(X, Y)$ represents an FD$\}$. Unmark edge $(X, Y)$,
          remove its supersets from $G$.
 6:    else
 7:       $O_X$ = subsumed $x_i$, $V_X = (X - O_X)$
 8:       findCFD$(O_X, X, Y, k)$
 9:       if $O_X \neq \emptyset$ then
10:          $G(X', Y', Q, P) = (V_X, O_X)$ {Generate next
             CFD candidate; add variable, conditional at-
             tributes. Mark edge $(X', Y')$.}
11:    if $k \geq 2$ and $G = \emptyset$ then
12:       break {No candidates remaining}
13: return $CL$ {List of conditional rules}

 DEFINE findCFD$(O_X, X, Y, k)$
 1: If $k \geq 2$: itn = 2, ELSE itn = 1
 2: for $i$: 1 to itn do
 3:    {First check for CFDs with added variables. If none
       found, check with added conditions.}
 4:    for $q_i$ in $Q$ do
 5:       if $q_i$ contains values only from $O_X$ then
 6:          if $M(x_i, y_i) \geq \tau$ then
 7:             $\varphi$: $X \rightarrow A$ {M: interest metric, $\tau$: threshold}
 8:             $CL = (CL \cup \varphi)$ {add discovered CFD to list}
 9:             $O_X = (O_X$ - ($x_i$ in $\varphi$)) {update classes}
10:    If CFDs found, break.
11: return $(O_X)$
```

3.4 Interest Measures for CFDs

To capture interesting rules we consider three interest measures, namely support, the χ^2-test, and conviction. We apply these metrics as a filter to avoid returning an unnecessarily large number of rules, and to prune rules that are not statistically significant nor meaningful.

3.4.1 Support

Support is a frequency measure based on the idea that values which co-occur together frequently have more evidence to substantiate that they are correlated and hence are more interesting. The support of a CFD φ: $(X \rightarrow A)$, *support*$(X \rightarrow A)$, is the proportion of tuples that match φ, defined as

$$S_\varphi = \frac{s = \#\ \text{tuples containing values in } X \text{ and } A}{N = \#\ \text{of tuples in} R} \tag{1}$$

For an attribute set X, *support*(X) is defined as the number of tuples containing values in X divided by N. If φ holds, then $s = \Sigma|y_i|$, for y_i that subsume x_i in Ω_X. For example, the rule φ : [SAL = '≤50K', REL] → [MR] holds for x_6, x_7 in $\Omega_{(SAL,REL)}$, which are subsumed by y_6, y_7 in Y. Hence, $s = |y_6| + |y_7| = 2 + 2 = 4$, giving $S_\varphi = (4/11)$. In our algorithm, we only consider rules that exceed a minimum support threshold θ, that is, we are interested in rules with $S_\varphi \geq \theta$.

3.4.2 χ^2-Test

Given $\varphi : X \rightarrow A$, we expect the support of φ to be the frequency in which the antecedent and consequent values occur together, i.e., (*support*(X)∗*support*(A)). If the actual support *support*$(X \rightarrow A)$ deviates significantly from this expected value, then the independence assumption between X and A is not satisfied. Rules with *support*$(X \rightarrow A) \approx$ (*support*(X) ∗ *support*(A)) are not interesting. We capture this idea by using the χ^2 test

$$\chi^2 = \frac{(E(v_X v_A) - O(v_X v_A))^2}{E(v_X v_A)} + \frac{(E(\bar{v_X} v_A) - O(\bar{v_X} v_A))^2}{E(\bar{v_X} v_A)} + \frac{(E(v_X \bar{v_A}) - O(v_X \bar{v_A}))^2}{E(v_X \bar{v_A})} + \frac{(E(\bar{v_X}\bar{v_A}) - O(\bar{v_X}\bar{v_A}))^2}{E(\bar{v_X}\bar{v_A})} \tag{2}$$

Let v_X, v_A be the set of constant values in φ from X and A, respectively. $O(v_X v_A)$ represents the number of tuples containing both v_X and v_A (including conditional values). $E(v_X v_A)$ is the expected number of tuples containing v_X and v_A under independence, that is, $E(v_X v_A) = \frac{O(v_X)O(v_A)}{N}$, where $O(v_X)$ is the number of tuples where v_X occurs. Similarly, $O(\bar{v_X})$ is the number of tuples where v_X does not occur. The χ^2 value follows a χ^2 distribution. We test for statistical significance by comparing the χ^2 value to critical values from standard χ^2 distribution tables. For a critical value of 3.84, only in 5% of the cases are the variables independent if χ^2 is greater than 3.84. Hence, if $\chi^2 > 3.84$ then the values $X \rightarrow A$ under Q are correlated at the 95% confidence level. We apply the χ^2 test both on its own and in conjunction with support to filter rules that are not statistically significant according to a given significance level.

3.4.3 Conviction

Various measures have been proposed to quantify meaningful attribute relationships and to capture interesting rules. Confidence is one of these measures. The confidence of φ: $(X \rightarrow A)$ can be stated as

$$\texttt{Confidence} = Cf_\varphi = \frac{Pr(X, A)}{Pr(X)} = \frac{Pr(Q, P, A)}{Pr(Q, P)} \tag{3}$$

where $X = (Q \cup P)$ and $Pr(Q, P)$ is equal to *support*(Q, P). Confidence measures the likelihood that A occurs given P under condition Q. We can compute this value by taking the number of occurrences of P and A (with Q) and dividing this by the number of occurrences of P (with Q). If P always occurs with A then the confidence value is 1. If they are independent, then confidence is 0. A drawback of confidence is that it does not consider the consequent A, and it is possible that $\frac{Pr(Q,P,A)}{Pr(Q,P)} = Pr(Q, A)$, indicating that P and A are independent. If this value is large enough to satisfy the desired confidence threshold, then an uninteresting rule is generated.

Interest is an alternative correlation metric that measures how much two variables deviate from independence. The interest of φ quantifies the deviation of P and A (with condition Q) from independence. We define interest for φ as

$$\texttt{Interest} = I_\varphi = \frac{Pr(Q, P, A)}{Pr(Q, P)Pr(Q, A)} \tag{4}$$

The value I_φ ranges from [0, ∞]. If $I_\varphi > 1$ this indicates that knowing P increases the probability that A occurs (under condition Q) . If P and A are independent then $I_\varphi = 1$. Although I_φ considers both $Pr(Q, P)$ and $Pr(Q, A)$ it is a symmetric measure, that fails to capture the direction of the implication in φ.

The asymmetric conviction metric [7] addresses this shortcoming. It is similar to interest but considers the directional relationship between the attributes. The conviction for φ is defined as

$$\texttt{Conviction} = C_\varphi = \frac{Pr(Q, P)Pr(\neg A)}{Pr(Q, P, \neg A)} \quad (5)$$

$\varphi : ((Q, P) \rightarrow A)$ can logically be expressed as $\neg((Q, P) \wedge \neg A)$, and conviction measures how much $((Q, P) \wedge \neg A)$ deviates from independence. When P and A are independent, conviction is equal to 1, and when they are related conviction has a maximal value approaching ∞. We apply these interest measures in our CFD validity test as shown in Algorithm 1. A qualitative study evaluating these measures is given in Section 5.2.

3.5 Pruning

We apply pruning rules to efficiently identify a set of minimal, non-redundant CFDs in R.

Conditioning Attributes. If a candidate $X \rightarrow A$ is a functional dependency over R, then all edges (XB, YB) representing supersets of edge (X, Y) are pruned. This is because $(X \cup B) \rightarrow A$ can be inferred from $X \rightarrow A$. Furthermore, supersets of $XA \rightarrow C$ for all other attributes C in R are also pruned, since we can remove A without affecting the validity of the dependency (whether it holds or not) and hence, is not minimal. For CFDs, we must take extra care since for each distinct conditional attribute value Q in $\varphi : X \rightarrow A$, we get different tuple values satisfying the rule (since φ holds only over a portion of the relation). If φ holds under condition Q, we prune x_i in Ω_X from being considered in subsequent evaluations of φ with conditions that are supersets of Q. This is to ensure that the discovered rules are minimal. For example, if $Q = U$, then $x_i \epsilon \Omega_X$ are not considered in subsequent evaluations involving $Q = UV, UVW, etc.$ They are however considered in rules with conditions not including U. We maintain a hash table that records which x_i to consider for a candidate rule under specific conditional attributes. The set Λ_X is updated to remove subsumed x_i as rules are discovered. For example, in Example 1 we will consider only x_4 and x_7 for candidates involving supersets of $(X, Y) = $ `((REL), (REL,MR))` with conditions that are supersets of `(REL)`.

Support Pruning. In support, we leverage the *anti-monotonic property* [3] that if *support*$(X) < \theta$, then any supersets of X will also fall below θ. If a set X is not large enough, then adding further attributes results in a new set that is either equal to or smaller than X. The first intuition is to apply this property to prune $|x_i| < \theta$. For example, suppose for candidate $(X, Y) = $ `((SAL,REL), (SAL,REL,MR))` in Example 2, $\theta = 0.3$. All the classes x_1, x_3, x_6 and x_7 have size equal to 2 and fall below θ, and hence should be pruned. However, if we were interested in only finding value based rules, this would work. But we are interested in finding rules that also contain variables. The rule φ: `[SAL = '≤50K', REL]` $\rightarrow$ `[MR]` holds under condition `SAL = '≤50K'` (class $x'_6 = \{6,7,8,10\}$ in $\Pi_{\texttt{SAL}}$) having support$(X \rightarrow A) = 4 \geq \theta$. We do not prune classes in X whose individual sizes are less than θ since the sum of these class sizes may satisfy θ under a common conditional value. Instead, we prune based on $|q_i|$ in Q. If the largest class in Q is less than θ, then clearly the current candidate rule does not contain sufficient support. Furthermore, any supersets of Q will also fall below θ. We maintain a list of these small conditional attribute sets and prune all candidate rules with conditions involving these small sets.

Frequency Statistics. When evaluating an edge (X, Y) in the first level of the lattice (where X and Y resolve to single value constants), we can leverage attribute value frequency statistics to prune CFD candidates in the reverse direction. Suppose we are considering a candidate φ: $[X =$ 'Toronto'$] \rightarrow [Y = 416]$. If the partition classes $|x_{Toronto}| = |y_{416}|$ then φ is a CFD. The value $|y_{416}|$ represents the co-occurrence frequency between the values '416' and 'Toronto'. The actual frequency count of value '416', f_{416}, may be larger than $|y_{416}|$, indicating that '416' co-occurs with other X values. This indicates that the reverse CFD candidate $[Y = 416] \rightarrow [X =$ 'Toronto'$]$ does not hold. More specifically, for a CFD $[X =$ 'x'$] \rightarrow [Y =$ 'a'$]$, where X is a single attribute, $Y = (X \cup A)$, if $(f_a \leq y_a)$ for some value 'a' in A, then the reverse CFD $[A =$ 'a'$] \rightarrow [X =$ 'x'$]$ holds. Otherwise, the y-class representative should be saved for further conditioning on multiple attributes. Note that we do not need to explicitly store the f_a values since these are the class partition sizes for the single attribute A. This process allows us to evaluate two CFD candidates ($A \rightarrow B$, $B \rightarrow A$) for the price of one, and we are able to prune half of the level-1 candidates, saving considerable computational effort.

4. IDENTIFYING DIRTY DATA VALUES

Real data often contains inconsistent and erroneous values. Identifying these values is a fundamental step in the data cleaning process. A data value can be considered dirty if it violates an explicit statement of its correct value. Alternatively, if no such statement exists, a value can be considered dirty if it does not conform to the majority of values in its domain. In particular, if there is sufficient evidence in the data indicating correlations among attribute values, then any data values participating in similar relationships, but containing different values are potentially dirty. We formalize this intuition and search for approximate CFDs over R. If such approximate rules exist, then there are records that do not conform to the rules and contain potentially dirty data values. We report these non-conformant values along with the rules that they violate. We define what it means for a value to be dirty using the support and conviction measures.

4.1 Using Support

A conditional rule $\varphi : X \rightarrow A$ is true if there is some condition q that holds for x_i in Ω_X. The remaining non-subsumed classes x_i in Λ_X, are not considered during the conditioning. Each of these x_i maps to two or more classes y_k in Π_Y indicating that the X value occurs with different Y values. Let Υ_Y be the set of $y_k \epsilon \Pi_Y$ that $x_i \epsilon \Lambda_X$ maps to. Let y_{max} represent the largest y_k class, and recall that θ is the minimum support level. If $|y_{max}| \geq (\theta * N)$ and $\exists y_s$ such that $|y_s| \leq (\alpha * N)$, $y_s \neq y_{max}$, then $\psi = ([X = v_i] \rightarrow [Y = v_s])$ is reported as a violation. The parameter α is an upper error threshold defining the maximum frequency (percentage) of a dirty data value. Similar thresholds have been used to find approximate FDs, where the threshold defines the maximum number of violating tuples allowed in order for an FD to qualify as approximate [13].

Intuitively, since there is no 1-1 mapping between any of the x_i in Λ_X and a specific y_k in Υ_Y, a CFD cannot hold over these values. However, we can check if there is a y_{max} that satisfies θ, and if so, then this is an approximate CFD (i.e., if the remaining y_k classes did not exist then ($[X = v_i] \rightarrow [Y = v_{max}]$ would be a CFD). We select y_{max} to be the largest class. The remaining v_k values co-occur with v_i at different frequencies, and if there are v_k values that occur *infrequently* (less than α), then they are inconsistent with respect to v_{max}. Once a set of dirty data values is found for a candidate (X, Y), the corresponding classes x_i are not considered again (for dirty values) in rules involving supersets of (X, Y). This is to avoid returning redundant dirty values from the same record.

EXAMPLE 3. *Returning to our Census Table 1, let $\theta = 0.15$ and $\alpha = 0.1$. We remarked that tuples t_3, t_6 and t_9 contained dirty values and we show how these values can be identified. Consider the following attribute partitions:*

- $\Pi_{(M,G)} = \{\{1,2,3\},\{5\},\{4,6,9\},\{10\},\{7\},\{8,11\}\}$
- $\Pi_{(M,G,R)} = \{\{1,2\},\{4,6\},\{3\},\{5\},\{7\},\{8\},\{9\},\{10\},\{11\}\}$
- $\Pi_{OCC} = \{\{1,3\},\{2,9\},\{4,6,10\},\{5,8\},\{7\},\{11\}\}$
- $\Pi_{CLS} = \{\{1,2\},\{3,5,7,11\},\{4,10\},\{6\},\{8\},\{9\}\}$
- $\Pi_{(C,O)} = \{\{1\},\{2\},\{3\},\{5\},\{4,10\},\{6\},\{7\},\{8\},\{9\},\{11\}\}$

Consider candidate (X, Y) = ((MR, GEN), (MR, GEN, REL)) and class $x_1 = (1, 2, 3)$, which maps to $y_1 = (1, 2)$ and $y_3 = (3)$. Since $y_{max} = y_1$ and $\frac{|y_1|}{N} \geq \theta$, and $y_s = y_3$, $\frac{|y_3|}{N} \leq \alpha$, we have a dirty value in tuple t_3. That is, [MR = 'Married', GEN = 'Male'] $\rightarrow$ [REL = 'Wife'] is inconsistent with the rule [MR = 'Married', GEN = 'Male'] $\rightarrow$ [REL = 'Husband'].

For candidate (X, Y) = ((OCC), (OCC, CLS)), $x_4 = (4, 6, 10)$ maps to $y_4 = (4, 10)$ and $y_6 = (6)$. Similar to above, $y_{max} = y_4$, $y_s = y_6$, and we have a non-conformant tuple t_6 with dirty values [OCC = '?'] $\rightarrow$ [CLS = 'Never-worked'] violating [OCC = '?'] $\rightarrow$ [CLS = '?'] with support = (2/11). Similarly, we discover the following inconsistent values in t_9: [REL = 'Own-child'] $\rightarrow$ [MR = 'Divorced'] and [REL = 'Own-child'] $\rightarrow$ [SAL = '$> 50K$'] which violate φ_1:[REL = 'Own-child'] $\rightarrow$ [MR = 'Never-married'] and φ_2:[REL = 'Own-child'] $\rightarrow$ [SAL = '$\leq 50K$'], respectively.

Dirty data values can prevent underlying rules from being discovered and being reported concisely. A larger set of CFDs are generated to get around the exception. For example, if a record contains ['Husband'] $\rightarrow$ ['Female'], the actual CFD ['Husband'] $\rightarrow$ ['Male'] does not hold, and an increased number of rules such as ['Husband', 'Doctorate'] $\rightarrow$ ['Male'] and ['Husband', 'Exec-mgr'] $\rightarrow$ ['Male'] are found. Our data quality tool is able to identify exceptions along with the potential (and violated) CFD and present these to an analyst for cleaning.

4.2 Using Conviction

We use the conviction measure to identify data values that are independent and violate an approximate CFD. Similar to the support case, we test if an approximate CFD φ : ($[X = v_i] \rightarrow [Y = v_{max}]$) satisfies a minimum conviction level C. We sort $|y_k|$ in Υ_Y in descending order to find rules containing the largest conviction value first. Let y_{max} represent the satisfying y_k with the largest conviction value (C_φ). To identify dirty records, we are interested in finding all tuples that have independent attribute values relative to φ. This means that for the y_k, if their conviction value $C_\sigma < C_\varphi$, for σ: ($[X = v_i] \rightarrow [Y = v_k]$), then v_k has less dependence with v_i than v_{max}, and hence is reported as an exception. We rank these exceptions according to descending ($C_\varphi - C_\sigma$) to find those values that deviate most from φ. Algorithm 2 presents pseudocode for the algorithm to identify dirty data values in R.

Algorithm 2 Algorithm to discover dirty data values

```
INPUT Relation $R$
1: Initialize $DL = \{\}$ dirty list
2: $multiSize(x_i)$ = all $y_i \subseteq x_i$
3: for each $x_i$ in $multiSize$ do
4:    $Xbench = 0$
5:    for $y_i$ in $multiSize(x_i)$ do
6:       $y_{max} = y_i$ with largest measure $M(x_i, y_i)$
7:       if ($M(x_i, y_{max}) < \tau$) and ($Xbench == 0$) then
8:          break {unsatisfied threshold, consider next $x_i$}
9:       else
10:         if ($Xbench == 0$) then
11:            $Xbench = 1$ {approximate CFD}
12:            $\varphi$: $[X$ = value of $x_i] \rightarrow [Y$ = value of $y_{max}]$
13:         else
14:            if $M(x_i, y_i) \leq \alpha$ then
15:               {$\alpha$: dirty threshold, $\delta$: dirty data value}
16:               $\delta : [X$ = value of $x_i] \rightarrow [Y$ = value of $y_i]$
17:               $DL = (DL \cup (\delta, \varphi))$ {save $\delta$ and violated rule
                  $\varphi$}
18: return $DL$
```

5. EXPERIMENTAL EVALUATION

We ran experiments to determine the effectiveness of our proposed techniques. We report our results using seven datasets (six real and one synthetic) and provide examples demonstrating the quality of our discovered rules and exceptions.

Real Datasets. We use six real datasets from the UCI Machine Learning Repository [4], namely the Adult, Census-Income (KDD), Auto, Mushroom, Statlog German Credit, and Insurance Company Benchmark datasets. The Adult dataset from the 1994/95 US Census contains 32561 records of US citizens describing attributes related to their salary (above or below \$50K). The Census-Income dataset is a larger dataset (similar to Adult) containing 300K tuples (299285 from the original dataset plus we append extra records) with 347 domain values. The Auto dataset describes characteristics of automobiles used for insurance risk prediction, with 205 tuples and 10 attributes. The Mushroom dataset describes physical characteristics of mushrooms and classifies them as either edible or poisonous and contains

Table 3: Experimental Parameters

Parameter	Description	Values
n	number of attributes in R	[8,17]
N	number of tuples in R	[25K, 300K]
d	attribute domain size	[270, 2070]
θ	minimum support level	[0.05, 0.5]

8124 tuples with 10 attributes. The Statlog German Credit dataset contains credit information for 1000 customers and rates them with either good/bad credit. Lastly, the Insurance Company Benchmark contains 9822 records describing product usage and socio-demographic information on customers from a real insurance company. We use these datasets in our qualitative study described in Section 5.2.

Synthetic Tax Dataset. This is a synthetic dataset consisting of individual US tax records. The data is populated using real data describing geographic zip codes, area codes, states, and cities, all representing real semantic relationships among these attributes. Furthermore, real tax rates, tax exemptions (based on marital status, number of children), and corresponding salary brackets for each state are used to populate the tax records. We use this dataset to test the scalability of our techniques.

Parameters. Our experiments were run using a Dual Core AMD Opteron Processor 270 (2GHz) with 6GB of memory. The parameters we evaluate are shown in Table 3. We use the χ^2 critical value equal to 3.84, representing a 95% confidence level. We set the minimum conviction level $C = 10$ to ensure that a sufficient number of rules are returned. We explored different values of θ and α and their individual effect on the discovery time. We observed experimentally (using the Adult dataset) that varying θ (and α) had a minimal effect on the running time. We tested this for each $n = [5, 9]$ and the running times did not vary significantly. For the qualitative study in Section 5.2, we set confidence $Cf = 0.5$, and interest $I = 2.0$ to capture rules that include dependent attributes.

5.1 Scalability Experiments

We used the Tax dataset and varied the parameter of interest to test its effect on the discovery running time.

Scalability in N. We studied the scalability of the discovery algorithm (with and without identifying exceptions) with respect to N. We considered the support, χ^2-test, and conviction measures. We fixed $n = 8, \theta = 0.5, \alpha = 0.01$ and $d = 173$. We ran the discovery algorithm for N ranging from 25K to 200K tuples. Figure 3 shows the results. The algorithm scales linearly for all the metrics considered. Support and conviction showed the best results (with support marginally outperforming conviction). There is a small overhead for computing exceptions. The results with the χ^2-test did not prune as many rules as expected (for $N =$ 25K, over 234 rules were returned). Furthermore, the rules with the highest χ^2 value were often ones that occurred least frequently. This lead us to consider support and the χ^2-test together as a measure to capture rules that are both statistically significant and that have a minimum frequency to substantiate their usefulness. Using the support with the χ^2-test returned the same number of rules as using support alone, but the χ^2-test ranks the set of rules satisfying θ. Since the number of rules returned is equal to the support

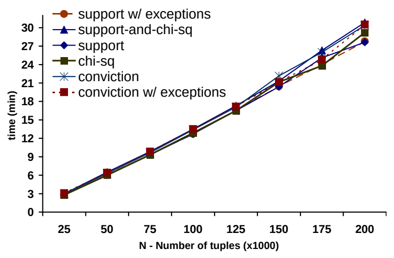

Figure 3: Scalability in N (Tax dataset)

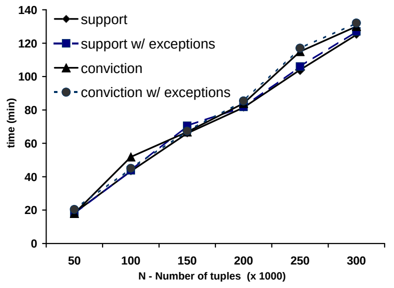

Figure 4: Scalability in N (Census-Income dataset)

measure, we only consider the support measure in subsequent tests. Figure 4 shows the running times using the real Census-Income dataset with a linear scale-up wrt N up to 300K tuples for $n = 8$. In both the Tax and Census-Income datasets, the support based measure performs slightly better than conviction particularly for larger N.

Scalability in n. We studied the impact of n on the discovery times. We considered $N = 30K, \theta = 0.5, \alpha = 0.01$ and d in [12,31]. Figure 5 shows that the number of attributes does affect the running time. This is due to the increased number of possible attribute sets X that need to be evaluated. An increased number of attributes causes a larger number of conditions and variables to be evaluated for a candidate rule. For smaller $n < 15$, we observe that the overhead of computing exceptions remains small for both the support and conviction measures. For larger n, this overhead increases more rapidly as there are more potential dirty values to evaluate. For each potential exception, we verify that it has not been considered previously (minimal) and that it satisfies the α threshold. For wide relations ($n > 15$), vertical partitioning of the relation into smaller tables can help reduce the time for finding exception tuples. However, for relations with up to $n = 16$ attributes, the discovery times for rules and dirty values are found reasonably in about 13 and 30 min respectively.

Scalability in d. We evaluate the effect of the domain size d on the discovery time. We fix $N = 30K, n = 6, \theta = 0.5$ and $\alpha = 0.01$ and varied d from 270 to 2070. Figure 6 shows the results using support and conviction. We expect d to affect the discovery times since an increase in d leads to a larger number of equivalence classes that must be compared. The running times using the support measure with and without exceptions are similar, and show a gradual increase. The execution times using conviction are slower, and there is an average overhead of about 40% for finding exceptions. This is likely due to weaker pruning strategies for conviction. In support, we prune all supersets of condi-

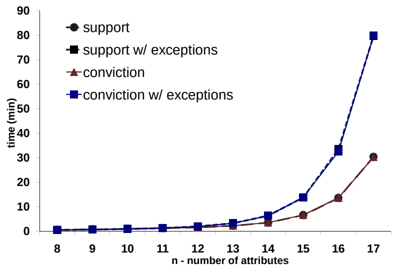

Figure 5: Scalability in n

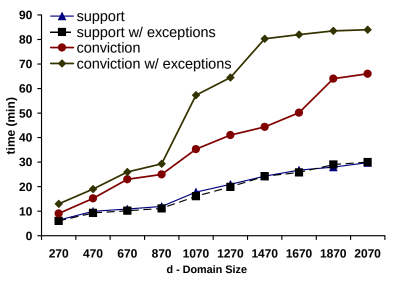

Figure 6: Scalability in d

tional attributes that do not satisfy the minimum support level (this property unfortunately does not hold when using conviction). This pruning can have a considerable benefit for increasing d as the number of classes increases. One solution to this problem is to group (stratify) the values (when d is excessively large) to reduce the domain size, where the stratification may be dataset dependent.

5.2 Qualitative Evaluation

We evaluate the quality of the discovered conditional rules and exceptions using the real datasets. Specifically, we consider the top-k discovered rules using support, the χ^2-test, support with χ^2-test, conviction, confidence and interest.

Data cleaning is subjective and we do not have a gold standard containing a *complete* set of cleaning rules. Hence, we do not report the recall of our approach. However, we can evaluate the rules that are returned based on their usefulness. Precision is a measure used in information retrieval to measure the percentage of documents retrieved that are relevant to a user's query. For each top-k set of returned rules, we compute precision to evaluate the effectiveness of each measure in returning rules that are interesting and that are useful towards data cleaning. We do this by manually evaluating each of the top-k rules and selecting those that are most relevant and that convey interesting information.[4] We compute the precision for the top-k rules for $k = 20$. This is also known as the *precision at* k, defined as

$$\text{Precision} = \frac{\#\ \text{relevant rules}}{\#\ \text{returned rules}} \tag{6}$$

In cases where less than k rules are returned, we set k to the number of returned rules. Table 4 shows the precision for each metric and dataset, along with the total number of rules returned in parentheses. Conviction is the superior measure providing the most interesting rules (across all datasets) followed by confidence, and with support and interest performing similarly. Conviction addresses the limitations of confidence, and interest and support, respectively, by considering the consequent and the direction of the implication. Both these characteristics are important for identifying meaningful semantic relationships. The German Credit dataset returned rules containing specific credit amounts that were not meaningful, hence giving lower precision values.

We found that the χ^2-test produced an unmanageably large number of rules, and rules with high χ^2 values occurred infrequently and were not interesting. This made it difficult to decipher which rules were most important, and manually evaluating each rule was not feasible. Instead, we used the χ^2-test (with support) to find rules that are statistically significant after satisfying a minimum support level. This returned a more reasonable number of rules with precision results similar to support. A sample of the discovered rules are given in Table 5. The results show that our technique is able to capture a broad range of interesting rules that can be used to enforce semantic data consistency. This is particularly useful when rules are unknown or are outdated, and new rules are required to reflect current policies. Many of the rules in Table 5 reveal intuitive knowledge and

Table 5: Sample of discovered rules. (*) indicates the rule was also found using conviction.

Support
[OCC = '?'] $\rightarrow$ [IND = '?'] (*) Unknown occupation implies industry is also unknown
[ED = 'Children'] $\rightarrow$ [SAL = '$\leq 50K$'] (*) School aged children should make less than \$50K
[[BODY = 'hatchback', CYL] $\rightarrow$ [PRICE] For hatchbacks, num cylinders determines price
[[CLR = 'yellow', BRUISE, ATTACH] $\rightarrow$ [CLS] (*) Yellow mushrooms, bruising and gills imply if it's poisonous
[[PUR = 'education'] $\rightarrow$ [FGN = 'yes'] Education loans normally taken by foreign workers
Conviction
[RATE = 3] $\rightarrow$ [DRS = 2] Dangerous cars have two doors
[BAL = 'delay'] $\rightarrow$ [FGN = 'yes'] Foreign workers normally delay paying off credit
[BAL = '< 0', PUR = 'educ', AMT] $\rightarrow$ [RATE] For low, educ loans, amount determines credit rating
[S-TYPE = 'large,low class'] $\rightarrow$ [M-TYPE = 'adult fam'] Lower class families have more adults living together
Confidence
([SM=foul,fishy, CLR='buff'] $\rightarrow$ [CLS='poison'] (*) Buff colour mushrooms, foul or fishy smell, are poisonous
Interest
([MR = 'single', PROP = 'yes'] $\rightarrow$ [LIFE = 'minimal'] Single persons owning property have minimal life insurance
χ^2
([S-TYPE = '76-88% family'] $\rightarrow$ [LIFE = 'none'] 76-88% of average families have no life insurance
([ED = 'Masters', FAM] $\rightarrow$ [SAL] For Masters educated, family type determines salary range

[4]This follows a similar process of computing relevance and precision in information retrieval where a user study determines the relevance of returned documents for a query.

Table 4: Precision results for discovered rules and exceptions. The total total number of returned rules and exceptions are shown in parentheses.

Dataset	Support	χ^2	Supp-χ^2	Conviction	Confidence	Interest	Supp-Ex	Convic-Ex
Adult	0.46 (13)	0.15 (9082)	0.46 (13)	0.75 (22)	0.7 (25)	0.6 (25)	0.33 (5749)	0.76 (1456)
Census-Income	0.83 (6)	0.55 (86)	0.83 (6)	0.92 (12)	0.89 (9)	0.83 (6)	0.56 (264)	0.9 (3148)
Auto	0.5 (18)	0.2 (1498)	0.35 (18)	0.65 (57)	0.65 (161)	0.55 (397)	0.56 (61)	0.83 (765)
Mushroom	0.75 (58)	0.5 (919)	0.6 (57)	0.75 (117)	0.65 (68)	0.65 (32)	0.67 (198)	0.7 (780)
German Credit	0.37 (8)	0.0 (62005)	0.43 (7)	0.45 (9710)	0.45 (6237)	0.1 (3053)	0.76 (581)	0.83 (11415)
Insurance	0.625 (8)	0.4 (17716)	0.63 (8)	0.8 (112)	0.5 (43)	0.5 (355)	0.37 (451)	0.67 (7921)
Average	**0.59**	**0.3**	**0.55**	**0.72**	**0.64**	**0.54**	**0.54**	**0.78**

Table 6: Sample of dirty values and violated rules.

Support
`[REL = 'Husband']` $\rightarrow$ `[GEN = 'Female']` Violates: $\rightarrow$ `GEN = 'Male'` (*)
`[SH = 'Convex', SM = 'none']` $\rightarrow$ `[CLS = 'edible']` Violates: $\rightarrow$ `CLS = 'poison'` (*)
`[REL = 'Own-child']` $\rightarrow$ `[SAL = '>50K']` Violates: $\rightarrow$ `[SAL = '≤50K']` (*)
Conviction
`[MR='single-male', PROP='rent']` $\rightarrow$ `[RATE='good']` Violates: $\rightarrow$ `RATE = 'bad'`
`[EMP='<1yr', PROP=?]` $\rightarrow$ `[RATE='good']` Violates: $\rightarrow$ `RATE = 'bad'`
`[CTRY = 'China']` $\rightarrow$ `[RACE = 'White']` Violates: $\rightarrow$ `RACE = 'Asian'`
`[S-TYPE='affluent young family']` $\rightarrow$ `[AGE='40-50']` Violates: $\rightarrow$ `AGE = '30-40'`

semantic patterns in the data.

Exception Values. We evaluated the quality of the identified exceptions (dirty values) using the support and conviction measures. Our definition of an exception using support is based on values that occur *infrequently.* Hence, we compute the *inverse of support*, and consider precision for the top-k violations returned where $k = 30$. Table 4 ('Ex' columns) shows the precision values, with the total number of dirty values returned in parentheses. Conviction clearly outperforms support and does a better job at capturing semantic deviations. Support returns values that may occur infrequently but are not necessarily incorrect. By considering both the magnitude of deviation from independence, and the direction, conviction captures intuitive, dirty values. A sample of the identified exceptions are given in Table 6.

The examples highlight that we are able to identify data values that are instinctively anomalous. Capturing dirty data values is important to avoid making poor, costly decisions. For example, the incorrect credit rating of 'good' to a renter and a person employed less than a year, classifying a type of mushroom as edible when it is poisonous, and incorrectly categorizing a young family into an older age group for a life insurance policy.

5.3 Comparative Evaluation

There are no equivalent techniques for discovering conditional functional dependencies. However, there are algorithms for discovering functional dependencies (FD) (one of the best known being Tane [13]), and for discovering frequent association rules (the best known being the Apriori algorithm [3]). In relation to our work, Tane discovers CFDs that have only variables so the rules must hold (or approximately hold) over the full relation. In contrast, an association rule is a CFD that contains only constants (not variables), and most association rule miners focus on rules with high support (frequent rules). Our work generalizes these techniques to find rules with both constants and variables. In our evaluation, we consider directly the quality of the rules we find compared with these two, more limited (but well-known), mining algorithms.

Given an error threshold ϵ, Tane searches for approximate FDs where the maximum percentage of tuples violating the FD is less than ϵ. We run Tane with $\epsilon = (1 - \theta)$. Apriori finds association rules between frequent itemsets by adding to large sets, and pruning small sets. We run Apriori using support and confidence. We use all the real datasets, and set $N = 100K, n = 7$ for the Census-Income dataset.

For $\epsilon = (1 - \theta)$, where θ ranges between [0.05, 0.2], Tane returned zero dependencies across all datasets. Since Tane does not search for conditional values where partial dependencies may hold, it prunes candidates more aggressively and performs poorly for high $\epsilon > 0.5$ (low θ support). Given these negative results, we ran Tane again using $\epsilon = \theta = 0.5$, and 4-15 approximate FDs were returned across all datasets except the German Credit dataset (104 approximate FDs).

Precision results are based on identifying dependencies that convey potentially meaningful attribute relationships (not arbitrary associations involving unique attribute values). The precision results are given in Table 7, and consider the top-20 dependencies. The CFD precision values show that informative rules are consistently identified over Tane. For example, in the mushroom dataset, Tane discovers an approximate dependency of (`[BRUISE, SHAPE]` $\rightarrow$ `[CLR]`) with a support of 50%, and while this is interesting, we are able to discover a more specific and accurate dependency (`[CLR = 'red', SPACE, BRUISE]` $\rightarrow$ `[CLS]`) with support 21% that Tane was unable to find. The running times for Tane were in the order of a few seconds to less than a minute. Although our discovery times are longer (in the range of 3-14 min) the performance overhead is dominated by the larger search space. The conditional dependencies we discover are more expressive than FDs and reveal specific semantic relationships that FDs are not able to capture.

We ran the Apriori algorithm with the corresponding θ support level from CFD discovery, and with confidence equal to 0.5. The Apriori algorithm runs quickly (with similar performance as Tane). However, it returns an unmanageably

Table 7: Comparative precision results between CFD, Tane (TNE), and Apriori (APRI) algorithms.

	$\epsilon \geq 0.8$			$\theta = \epsilon = 0.5$		Conf = 0.5	
Data	CFD	TNE	APRI	CFD	TNE	CFD	APRI
Adult	0.46	0	0.1	0.53	0.67	0.7	0.16
Census	0.83	0	0.4	0.83	0.5	0.88	0.4
Auto	0.5	0	0.23	0.82	1	0.65	0.46
Mushrm	0.75	0	0.43	1	1	0.65	0.3
Credit	0.375	0	0.5	0.875	0.07	0.45	0.5
Insure	0.625	0	0.5	0.625	0.57	0.5	0.3

large number of rules. Many of these rules are redundant (i.e., extensions of simpler rules with attributes added to the antecedent), and many rules can be inferred from other rules. Apriori returned [408, 24103] and [569, 34159] association rules using support and confidence, respectively. Our CFD discovery algorithm identified [8, 58] rules using support, and [9, 6237] rules using confidence; a more reasonable number of rules to manage. Although there are techniques for pruning and aggregating association rules, such as using closed frequent itemsets [25], seeking highly correlated itemsets [8], and dimension hierarchies, these techniques can still return a larger number of rules than our method. For example, in the Mushroom dataset at $\theta = 0.1$, Apriori returns 24103 rules, the closed itemset method returns 1197 rules at $\theta = 0.2$ [25], whereas our CFD discovery algorithm returns only 58 rules at $\theta = 0.1$, a reduction of 415 and 20 times, respectively. Furthermore, in addition to supporting variable based dependencies, CFDs are a more informative model when the goal is data cleaning rather than general modeling of frequent patterns. The precision results in Table 7 show that our CFDs provide more meaningful and understandable rules using either support or confidence.

Our discovery algorithm helps to bridge the gap between variable dependencies and value based association rules. The experimental results show that our techniques are able to capture interesting semantic rules to enforce a broad range of domain constraints, and for identifying data values that do not conform to these constraints.

6. CONCLUSIONS

We have presented a data-driven tool that discovers conditional functional dependencies which hold over a given data instance. These rules are useful in data cleaning and towards enforcing semantic data consistency. Furthermore, to help identify and correct dirty data values, we presented an algorithm that searches for approximate conditional rules and identifies exceptions to these rules, which are potentially dirty. Identifying both erroneous values and contextual data quality rules is a primary task towards improved data quality. Our qualitative and comparative studies (using various interest measures) demonstrated the effectiveness of our techniques for discovering a reasonable number of rules and those that are most interesting.

Acknowledgments. We thank Tasos Kementsietsidis and Xibei Jia for providing us with the tax data generator.

7. REFERENCES

[1] C. Aggarwal and P. Yu. Mining large itemsets for association rules. *Data Eng Bull*, 21(1):23–31, 1998.

[2] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93*, pages 207–216, 1993.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB '94*, pages 487–499, 1994.

[4] A. Asuncion and D. Newman. UCI ML repository, 2007.

[5] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE '07*, pages 746–755, 2007.

[6] L. Bravo, W. Fan, and S. Ma. Extending dependencies with conditions. In *VLDB '07*, pages 243–254, 2007.

[7] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD'97*, pages 255–264, 1997.

[8] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding interesting associations without support pruning. *IEEE TKDE*, 13(1):64–78, 2001.

[9] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB '07*, pages 315–326, 2007.

[10] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or how to build a data quality browser. In *SIGMOD '02*, pages 240–251.

[11] W. Eckerson. Achieving Business Success through a Commitment to High Quality Data. Technical report, Data Warehousing Institute, 2002.

[12] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB '01*, pages 371–380, 2001.

[13] Y. Huhtala, J. Kinen, P. Porkka, and H. Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *ICDE '98*, pages 392–401, 1998.

[14] H. V. Jagadish, J. Madar, and R. T. Ng. Semantic compression and pattern extraction with fascicles. In *VLDB '99*, pages 186–198, 1999.

[15] H. V. Jagadish, R. T. Ng, B. C. Ooi, and A. K. H. Tung. Itcompress: An iterative semantic compression algorithm. In *ICDE '04*, page 646, 2004.

[16] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1):129–149, 1995.

[17] L. V. S. Lakshmanan, R. T. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *SIGMOD '99*, pages 157–168.

[18] S. Lopes, J.-M. Petit, and L. Lakhal. Efficient discovery of functional dependencies and armstrong relations. In *EDBT '00*, pages 350–364, 2000.

[19] M. J. Maher. Constrained dependencies. *Theoretical Computer Science*, 173(1):113–149, 1997.

[20] J. Pei and J. Han. Constrained frequent pattern mining: a pattern-growth view. *SIGKDD Explor. '02*, 4(1):31–39.

[21] V. Raman and J. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB '01*, pages 381–390, 2001.

[22] I. Savnik and P. A. Flach. Discovery of multivalued dependencies from relations. *Intelligent Data Analysis*, 4(3,4):195–211, 2000.

[23] P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis, and T. Sellis. Arktos: towards the modeling, design, control and execution of etl processes. *Inf. Syst. '01*, 26(8):537–561.

[24] C. Wyss, C. Giannella, and E. L. Robertson. Fastfds: A heuristic-driven, depth-first alg for mining fds from relations. In *DaWaK '01*, pages 101–110, 2001.

[25] M. Zaki. Generating non-redundant association rules. In *KDD '00*, pages 34–43, 2000.