Data Cleaning: An Abstraction-based Approach

Dileep Kumar Koshley
Department of Computer Science and Engineering
Indian Institute of Technology Patna, India
Email: dileep.mtcs13@iitp.ac.in

Raju Halder
Department of Computer Science and Engineering
Indian Institute of Technology Patna, India
Email: halder@iitp.ac.in

Abstract—**Bertossi et al. proposed a data-cleaning technique based on matching dependences and matching functions, which is, in practice, intractable for some cases during the application of matching dependences in random orders. Moreover, the result of the application of a single matching dependence on a dirty database instance is a set of clean instances depending on the number of dirty tuples, which results a high computational overhead as well as large space requirement. The aim of this paper is to propose an improvement of the Bertossi's approach based on the Abstract Interpretation framework. This yields a single clean abstract database instance which is a sound approximation of all possible concrete clean instances. The convergence of the cleaning process can also be guaranteed by using widening operators in the abstract domain. The proposal improves significantly the efficiency and performance of the query systems** *w.r.t.* **the Bertossi's one.**

Keywords—Data Cleaning, Abstract Interpretation, Relational Databases

I. Introduction

The presence of bad data in databases may affect badly the quality of query answers in information processing systems. For instance, in case of decision making processes, the influence of incorrect or inconsistent data on the result of data-analysis may lead to a false decisive stand for an organization. Bad data may occur due to various reasons and falls into various categories: A list of 33 categories of dirty data is reported in [19]. Some of them are:

- **Data ambiguity**: Data ambiguity is a case where two separate pieces of data have common representation. For example, when two names *Frederick James Smith* and *Frederick John Smith* are represented as *Frederick J. Smith*.
- **Typo-errors**: It is common where data entry has to be done by humans. Bad data may occur due to typo-errors by users. For instance, if instead of "I.B.M.", the record is entered incorrectly as "I.B.N.".
- **Entity resolution problem**: This is not due to an error but is simply an artifact of having multiple ways to refer to the same real-world entity. For instance, if "I.B.M." is entered as "International Business Machine".
- **Measurement errors**: Errors in data may be due to the improper surveys or sampling strategies, e.g. in case of statistical survey like population counting.
- **Data integration errors**: Its common that a database contains information collected from multiple sources via multiple methods. Integration of multiple data sources may lead to errors.

Data cleaning is a promising field of research whose aim is to identify the presence of errors and inconsistencies in the data and to remove those in order to improve the quality of data analysis results. There has been a remarkable series of works in the literature, and a comprehensive survey on various data-cleaning problems and approaches at both schema and instance levels is reported in [22]. The existing works are categorized into dependence-based [3], [20], [5], [15], [6], probabilistic [1], holistic [8], adaptive [4], etc. However, most of the works suffer from the lack of efficiency, robustness and accuracy. The consistency and implication problems become undecidable in some approaches [5], [6].

A. Motivations and Contributions

The data cleaning approach by Bertossi et al. in [2] is based on the matching dependences (MDs) and matching functions. Application of single MD on tuples generates a set of clean instances corresponding to each pair of bad tuples. This increases the computational complexity of query answering as well as demands a large space requirement. Also the order of MDs sometime affects the cleaning process badly, making the process intractable. In practice, many organizations like health sector, census organizations, national security agencies etc. deal with large amount of information in their databases. Cleaning such large amount of information using Bertossi's method is, therefore, inefficient and may be impractical sometimes.

In this paper, we propose an improvement of the Bertossi's approach based on the Abstract Interpretation framework [10]. Abstract Interpretation theory is a semantics-based sound approximation method whose purpose is to over-approximate the dynamical behavioral properties of any computing systems. Our proposal results into a single clean abstract database instance which is a sound approximation of all possible concrete clean instances. The convergence of the cleaning process can also be guaranteed by applying widening operators in the abstract domain [9]. The proposal improves significantly the efficiency and performance of the query systems for large databases *w.r.t.* the Bertossi's one.

The structure of the paper is as follows: Section II discusses the related works in the literature. Section III recalls the basics of the Abstract Interpretation theory and the notion of matching depenedences and matching functions. Our proposed technique is discussed in section IV. We describe the computational complexity and space requirement of our approach in section V. An experimental evaluation is provided in section VI. Finally, we conclude our work in section VII.

II. Related Works

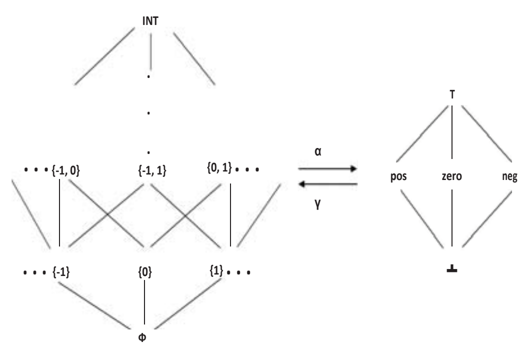
The existing approaches on data matching and cleaning are dependence based [3], [20], [5], [15], probabilistic [1], holistic [8], adaptive [4], etc. Authors in [3] defined Functional Dependences (FDs) as integrity constraints that encode data semantics. [5] is based on Conditional Functional Dependences (CFDs). A CFD is a functional dependence (FD) that holds on a subset of the relation specified in an accompanying pattern tableau. CFDs can express the semantics of data fundamental to data cleaning. Authors in [6] introduced a concept of Conditional Inclusion Dependences (CINDs) as an extension to traditional Inclusion Dependences (INDs). INDs associate attributes in a source schema with semantically related attributes in a target schema. CINDs enforces binding of semantically related data values across the source and target schema. Matching Dependences (MDs) were studied in [16], [2] as semantic constraints for data cleaning. However MDs in [16] do not specify how the matching of attribute values is to be done. Authors in [2] have introduced a process of cleaning an instance using matching dependences, as a chase-like procedure. Enforcing a matching dependence specifies that a pair of attribute values in two database tuples are to be matched, i.e. made equal, if similarities hold between other pairs of values in the same tuples. The similarity metrics to estimate the similarity between two values are explained in [14]. However, it is very inefficient in case of large databases with large number of functional dependences. Authors in [1] proposed a probabilistic approach that permits declarative query answering over duplicated data, where each duplicate is associated with a probability of being in the clean database. In [8], authors proposed a model that takes as input the denial constraints that are defined as declarative specification of the quality rules which generalize and enlarge the current class of constraints for cleaning data. Authors in [4] presented an adaptive approach of duplicate detection based on two similarity measures of strings: The first one utilizes the Expectation-Maximization (EM) algorithm based on string edit distance with affine gaps, whereas the other one employs a Support Vector Machine (SVM) to obtain a similarity estimate based on the vector-space model of text. In [13] authors presented a general framework consisting of six steps: selection of attributes, formation of tokens, selection of the clustering algorithm, similarity computation for the selected attributes, selection of the elimination function, and finally merge. [18] presented a solution to Merge/Purge problem where databases acquired from different sources are merged. A list of publicly available matching systems is provided in [7]. A commodity data cleaning system, NADEEF, was introduced in [12]. A recent survey by Köpcke and Rahm provides a comparative evaluation of several data matching systems [21].

III. Preliminaries

In this section, we recall some basics of the Abstract Interpretation Theory and their domains for strings and numerical values [11]. We also brief the notion of matching dependences and matching functions from [2].

A. Abstract Interpretation Theory

Abstract Interpretation is a sound approximation technique of the semantics of computing systems. The aim is to represent the concrete objects (e.g. variable values, object instances, traces etc.) by properties of interests representing abstract domains. The correspondence between the concrete semantics domain $\mathfrak{D}^c$ and the abstract semantics domain $\mathfrak{D}^a$ is formalized by Galois Connection $\langle \mathfrak{D}^c, \alpha, \gamma, \mathfrak{D}^a \rangle$, where α, γ are the abstraction and concretization functions respectively. Let `INT` be the set of integers and `SIGN` be the abstract domain of sign. The Galois connection $\langle \wp(\texttt{INT}), \alpha, \gamma, \texttt{SIGN} \rangle$ is represented below :



When either α is surjective or γ is injective, the Galois Connection $\langle \mathfrak{D}^c, \alpha, \beta, \mathfrak{D}^a \rangle$ is called Galois Insertion. In this case, $\alpha \circ \gamma = \lambda S^\sharp .\ S^\sharp$.

Various relational and non-relational abstract domains are proposed in the literature. For instance, the abstract domains for numerical values are the domain of sign, parity, intervals, polyhedra, octagons, etc [11]. The abstract domains for string values are character inclusions, bricks, string graphs, prefix and suffix, etc [10].

B. Matching Dependences and Matching Functions

Let `X` and `Y` be two sets of attributes. A matching dependence (denoted `X` $\rightarrow$ `Y`) specifies that the `Y`-values are to be matched, *i.e.*, are to be made equal, if similarity holds in `X`-values for the tuples in the database. Matching functions impose a lattice-theoretic structure on attribute domains. If two values are to be made equal, matching function produces a value that contains the information contained in both the values. For instance, consider the following database instance t_0. Let us assume a matching dependence which states that if for two tuples the names and phone numbers are similar, then their addresses must be similar (denoted {**Name, Phone**}→ **Address**). Matching function combines the information in those address values and generates t'_0, assuming that the values of **Name** and **Phone** in t_0 are similar.

Name	Phone	Address
John Doe	(613)12345	Main St, Ottawa
J Doe	12345	25 Main St

(a) Table t_0

Name	Phone	Address
John Doe	(613)12345	25 Main St, Ottawa
J Doe	12345	25 Main St, Ottawa

(b) Table t'_0 after applying matching function on **Address**

TABLE I: Application of matching dependence and matching function

IV. Proposed Approach

The main objective of our proposed approach is to generate an abstract clean instance which is a sound approximation of all possible concrete clean instances. To this aim, we apply the Abstract Interpretation framework combining with similarity measures. In particular, the proposed technique consists of the following four phases: Clustering, Abstraction, Application of MDs, and Sound Query Answering. The overall cleaning process is depicted in the form of flowchart in figure 1. Below we describe each of the phases in detail.

A. Clustering:

Similarity based matching dependence $\mathtt{X} \rightarrow \mathtt{Y}$ where $\mathtt{X}$ and $\mathtt{Y}$ are the sets of attributes, defines the data-semantics in relational database model [2]. This says that for any set of tuples, if the values of the attributes $\mathtt{X}$ are similar, then their $\mathtt{Y}$-values should be same.

As a first step, our approach uses a similarity metric to group the values into a set of clusters for attributes appearing on the left hand side of all the given MDs – values in a cluster are similar to each other. Various similarity metrics exist in the literature [7]. However, the choice of suitable clustering approach and similarity metric is out of the scope of this work.

Let t be a table where $\mathtt{attr}(t) = \{a_1, a_2, ..., a_n\}$. Let D_i be the domain of the attribute a_i. The values in t corresponding to the attribute a_i is:

$$\mathtt{Val}(a_i, t) = \pi_{a_i}(t) \subseteq D_i$$

where π represents projection operator. Applying similarity metric $\mathtt{F}$, we obtain a set of clusters

$$\mathtt{C}(a_i, t) = \mathtt{F} \circ \mathtt{Val}(a_i, t) = \{c_1, c_2, ..., c_m\}$$

Example 1. *To illustrate, let us consider the following table t_1 in Table II. Let us assume that, applying a suitable clustering algorithm*[1]*, we obtain the following clusters on the attributes Name, Address and Company:*

$$\begin{aligned}
\mathtt{C}(\textbf{\textit{Name}}, t_1) =& \{c_1^{name}, c_2^{name}, c_3^{name}\} \\
=& \{\textit{(Caeser Doe, C Doe), (Christian Doe, C Doe),} \\
& \textit{(Peter Christian, Peter Christ)}\} \\
\mathtt{C}(\textbf{\textit{Address}}, t_1) =& \{c_1^{addr}, c_2^{addr}, c_3^{addr}\} \\
=& \{\textit{(Main St Ottawa, Main St),} \\
& \textit{(25 Main St, Main St),} \\
& \textit{(25 MG Road Patna, MG Road Patna)}\} \\
\mathtt{C}(\textbf{\textit{Company}}, t_1) =& \{c_1^{comp}, c_2^{comp}, c_3^{comp}\} \\
=& \{\textit{(IBM India, IBM),} \\
& \textit{(IBM, IBM International),} \\
& \textit{(Samsung Electronics, Samsung)}\}
\end{aligned}$$

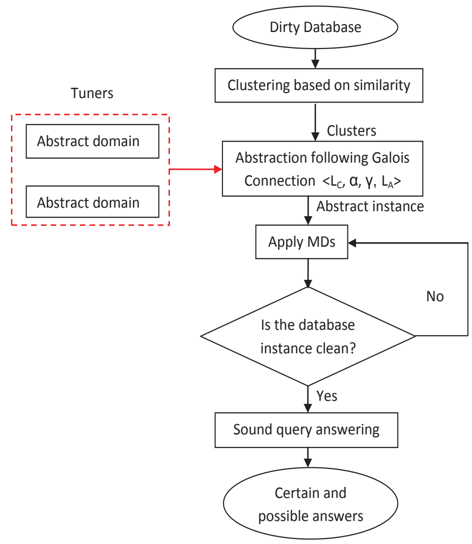


Fig. 1: Flowchart of the proposed approach

Name	Address	Company
Caeser Doe	Main St Ottawa	IBM India
C Doe	Main St	IBM
Christian Doe	25 Main St	IBM International
Peter Christian	25 MG Road Patna	Samsung Electronics
Peter Christ	MG Road Patna	Samsung

TABLE II: Table "t_1"

B. Abstraction:

In case of large database, application of MDs on each pair of bad tuples in a concrete database results into a large number of clean instances. This introduces a high computational overhead in query-processing system as well as large memory space requirement.

To cope with this, in this phase, we apply abstract domains aiming at replacing concrete values by suitable properties of interest, and we follow the Abstract Interpretation theory [11]. The aim is to generate an

[1]Note that the choice of suitable clustering algorithm is beyond the scope of the work.

abstract clean instance which is a sound approximation of all possible concrete clean instances, which reduces the computational complexity of query processing and the space requirement.

Various abstract domains exist, e.g. numerical concrete values can be abstracted by domain of intervals, domain of parity, domain of sign, etc. Similarly, string values can be abstracted using character inclusion, prefix and suffix, bricks, string graphs, etc [11], [10]. For cleaning dirty databases, we choose the domains of intervals and bricks as the suitable abstract domains for numerical and string values respectively.

Let $\mathbb{C}$ be the set of all clusters on a domain of values. We assume that the clusters may overlap each other. The powerset $\wp(\mathbb{C})$ forms a complete lattice $\mathtt{L_C} = (\wp(\mathbb{C}), \subseteq, \mathbb{C}, \emptyset, \cup, \cap)$. Consider a complete lattice $\mathtt{L_A} = (\mathbb{A}, \sqsubseteq, \top, \bot, \sqcup, \sqcap)$ of an abstract values domain representing the properties of concrete values. We define the Galois Connection between $\mathtt{L_C}$ and $\mathtt{L_A}$ as $\langle \mathtt{L_C}, \alpha, \gamma, \mathtt{L_A} \rangle$. Given a concrete value x, we define the abstraction function as follows:

$$\alpha(x) = \begin{cases} \alpha(\{c_1\} \cup \cdots \cup \{c_k\}) = \alpha(\{y \mid \exists i \in [1 \ldots k].\ y \in c_i\}) \\ \text{if } x \in c_1 \wedge \cdots \wedge x \in c_k \wedge \{c_1, \ldots, c_k\} \subseteq \mathbb{C} \\ \\ \bot \quad \text{if } x \text{ is NULL.} \end{cases}$$

That is, the abstraction of a concrete value is the abstraction of the lowest upper bound of all clusters in which x belongs. Similarly we can define the concretization function γ.

As we deal with relational database models, the tuple-wise abstraction is defined as $\alpha(\langle x_1, \ldots, x_n \rangle) = \langle \alpha(x_1), \ldots, \alpha(x_n) \rangle$.

Example 2. *Consider the concrete table t_1 in Table II. Following the Galois Connection defined above, we obtain the abstract version $t_1{}^\sharp$ corresponding to t_1 depicted in Table III.*

C. Application of MDs:

In this phase, we discuss the application of MDs and matching functions to clean dirty data in the abstract domain.

Let us first define matching functions on a set of abstract values in the domain of "Bricks" and "Intervals" respectively:

1) Numerical abstract values [11]: Given two intervals $[l_1, h_1]$ and $[l_2, h_2]$, the matching function is defined in terms of lattice least upper bound as follows:

$$\sqcup([l_1, h_1], [l_2, h_2]) = \begin{cases} [l_1, h_2] \text{ if } l_1 \le l_2 \text{ and } h_1 \le h_2. \\ [l_2, h_2] \text{ if } l_2 \le l_1 \text{ and } h_1 \le h_2. \\ [l_1, h_1] \text{ if } l_1 \le l_2 \text{ and } h_2 \le h_1. \\ [l_2, h_1] \text{ if } l_2 \le l_1 \text{ and } h_2 \le h_1. \end{cases}$$

Similarly, for sign and parity abstract domains, the matching functions are defined in terms of least upper bound of the corresponding lattices.

2) String bricks abstract values [10]: Given two abstract brick values $[S_1]^{m_1, M_1}$ and $[S_2]^{m_2, M_2}$, the matching function is defined as

$$\sqcup([S_1]^{m_1, M_1}, [S_2]^{m_2, M_2}) = [S_1 \cup S_2]^{min(m_1, m_2), max(M_1, M_2)}$$

The least upper bound operator on list of bricks is as follows: given two lists L_1 and L_2, we make them to have the same size n adding empty bricks ($[\emptyset]^{0,0}$) to the shorter one. Then the least upper bound is computed as:

$$\sqcup(L_1, L_2) = L_R[1] L_R[2] \ldots L_R[n]$$

where $\forall i \in [1, n] :\ L_R[i] = \sqcup(L_1[i], L_2[i])$.

We now discuss the application of a set of MDs for cleaning the dirty database.

The abstract values in "Bricks" domain are divided into two parts: *must*-part and *may*-part. For instance, {"Straw", "Straw Berry"} can be represented as a single Brick value $[\{''Straw''\}]^{1,1}[\{''Berry''\}]^{0,1}$. Here $[\{''Straw''\}]^{1,1}$ is the *must*-part and $[\{''Berry''\}]^{0,1}$ is the *may*-part.

We define matching dependence on an abstract table as follows: let $t^\#$ be an abstract dirty database. The matching dependence $\mathtt{X}^\sharp \rightarrow \mathtt{Y}^\sharp$ on $t^\sharp$ represents that if, for a set of tuples, the *must*-part of the abstract values corresponding to the attributes $\mathtt{X}^\sharp$ are same, then their $\mathtt{Y}^\sharp$-values must be same. Observe that, along with the *must*-part equality, we may also integrate some other similarity measures on *may*-part to identify the similarity in the abstract domain and to identify the applicability of MDs. This improves the preciseness of the clean results.

Let us illustrate the cleaning process in the abstract domain.

Example 3. *Consider the abstract table $t_1^\sharp$ in Table III(b). Consider the MD: ψ_1 which states that if for two tuples the name, address and company are similar, then their companies must be matched (deonted ψ_1 ={**Name, Address, Company**}→{**Company**}) .*

*As for the first three tuples the must-part of the abstract values in attribute **Name, Address, Company** are similar, we apply ψ_1 which cleans their companies using the matching function and makes them identical. The final clean abstract table $T^\#$ is shown in Table IV.*

Observe that the approach in [2] results into two different clean instances T_1 and T_2 depicted in table V, and the abstract clean instance $T^\#$ (in Table IV) is a sound approximation of the concrete clean instances (in Tables V(a) and V(b)), i.e. $T_1 \in \gamma(T^\#)$ and $T_2 \in \gamma(T^\#)$.

We now show another example where the order of MDs affects the cleaning result. We consider the following two scenarios:

Case-1: If an attribute lies on the left hand side of an MD, but not on the right hand side of any MDs, then we impose a restriction on the order of application of MDs starting with that MD. Let $A_2 \rightarrow A_3$, $A_1 \rightarrow A_2$, $A_3 \rightarrow A_4$ be a set of MDs. If we apply these MDs in any order,

$\mathbf{Name}^\sharp$	$\mathbf{Address}^\sharp$	$\mathbf{Company}^\sharp$
$\alpha(\{c_1^{name}\})$	$\alpha(\{c_1^{addr}\})$	$\alpha(\{c_1^{comp}\})$
$\alpha(\{c_1^{name}\},\{c_2^{name}\})$	$\alpha(\{c_1^{addr}\},\{c_2^{addr}\})$	$\alpha(\{c_1^{comp}\},\{c_2^{comp}\})$
$\alpha(\{c_2^{name}\})$	$\alpha(\{c_2^{addr}\})$	$\alpha(\{c_2^{comp}\})$
$\alpha(\{c_3^{name}\})$	$\alpha(\{c_3^{addr}\})$	$\alpha(\{c_3^{comp}\})$
$\alpha(\{c_3^{name}\})$	$\alpha(\{c_3^{addr}\})$	$\alpha(\{c_3^{comp}\})$

(a) Applying α

$\mathbf{Name}^\sharp$	$\mathbf{Address}^\sharp$	$\mathbf{Company}^\sharp$
$[C]^{1,1}[aeser]^{0,1}[Doe]^{1,1}$	$[Main]^{1,1}[St]^{1,1}[Ottawa]^{0,1}$	$[IBM]^{1,1}[India]^{0,1}$
$[C]^{1,1}[aeser, hristian]^{0,1}[Doe]^{1,1}$	$[25]^{0,1}[Main]^{1,1}[St]^{1,1}[Ottawa]^{0,1}$	$[IBM]^{1,1}[India, International]^{0,1}$
$[C]^{1,1}[hristian]^{0,1}[Doe]^{1,1}$	$[25]^{0,1}[Main]^{1,1}[St]^{1,1}$	$[IBM]^{1,1}[International]^{0,1}$
$[Peter]^{1,1}[Christ]^{1,1}[ian]^{0,1}$	$[25]^{0,1}[MG]^{1,1}[Road]^{1,1}[Patna]^{1,1}$	$[Samsung]^{1,1}[Electronics]^{0,1}$
$[Peter]^{1,1}[Christ]^{1,1}[ian]^{0,1}$	$[25]^{0,1}[MG]^{1,1}[Road]^{1,1}[Patna]^{1,1}$	$[Samsung]^{1,1}[Electronics]^{0,1}$

(b) Abstract Values using "Bricks" Domain

TABLE III: Abstract table ${t_1}^\sharp$

$\mathbf{Name}^\sharp$	$\mathbf{Address}^\sharp$	$\mathbf{Company}^\sharp$
$[C]^{1,1}[aeser]^{0,1}[Doe]^{1,1}$	$[Main]^{1,1}[St]^{1,1}[Ottawa]^{0,1}$	$[IBM]^{1,1}[India, International]^{0,1}$
$[C]^{1,1}[aeser, hristian]^{0,1}[Doe]^{1,1}$	$[25]^{0,1}[Main]^{1,1}[St]^{1,1}[Ottawa]^{0,1}$	$[IBM]^{1,1}[India, International]^{0,1}$
$[C]^{1,1}[hristian]^{0,1}[Doe]^{1,1}$	$[25]^{0,1}[Main]^{1,1}[St]^{1,1}$	$[IBM]^{1,1}[India, International]^{0,1}$
$[Peter]^{1,1}[Christ]^{1,1}[ian]^{0,1}$	$[25]^{0,1}[MG]^{1,1}[Road]^{1,1}[Patna]^{1,1}$	$[Samsung]^{1,1}[Electronics]^{0,1}$
$[Peter]^{1,1}[Christ]^{1,1}[ian]^{0,1}$	$[25]^{0,1}[MG]^{1,1}[Road]^{1,1}[Patna]^{1,1}$	$[Samsung]^{1,1}[Electronics]^{0,1}$

TABLE IV: $\mathtt{T}^\#$ - Abstract clean instance after applying ψ_1 on $t_1^\sharp$

Name	**Address**	**Company**
Caeser Doe	Main St Ottawa	IBM India
C Doe	Main St	IBM India
Christian Doe	25 Main St	IBM International
Peter Christian	25 MG Road Patna	Samsung Electronics
Peter Christ	MG Road Patna	Samsung Electronics

(a) $\mathtt{T}_1$ - Concrete clean instance-1 after applying ψ_1 on t_1

Name	**Address**	**Company**
Caeser Doe	Main St Ottawa	IBM India
C Doe	Main St	IBM International
Christian Doe	25 Main St	IBM International
Peter Christian	25 MG Road Patna	Samsung Electronics
Peter Christ	MG Road Patna	Samsung Electronics

(b) $\mathtt{T}_2$ - Concrete clean instance-2 after applying ψ_1 on t_1

TABLE V: Concrete clean instances after applying the approach in [2]

the result may not be correct. For instance, if we apply $A_3 \rightarrow A_4$ first, it means we are cleaning A_4 values based on A_3 values. However, as A_3 values may be dirty, this results a wrong clean instance. Therefore, before applying $A_3 \rightarrow A_4$, we have to clean A_3 by applying $A_2 \rightarrow A_3$. Similarly, before applying $A_2 \rightarrow A_3$, we have to clean A_2 by applying $A_1 \rightarrow A_2$.

Therefore the correct order is $A_1 \rightarrow A_2$, $A_2 \rightarrow A_3$, $A_3 \rightarrow A_4$.

Example 4. *Let us consider the database instance t_2 in Table VI.*

Manager	**Project name**	**Location**
Dileep	Flyover Traffic Decongestion	25 MG Road Patna
Dileep Kumar	Traffic Decongestion	MG Road Patna
Aman Kumar	E Traffic Decongestion	Patna

TABLE VI: Table t_2

*Suppose, the clusters on attribute values of **Manager** and **Project name** are as follows:*

$$\begin{aligned} \mathtt{C}(\textbf{\textit{Manager}}, t_2) &= \{c_1^{mang}, c_2^{mang}\} \\ &= \{\textit{(Dileep, Dileep Kumar),} \\ &\quad \textit{(Aman Kumar)}\} \\ \mathtt{C}(\textbf{\textit{Project name}}, t_2) &= \{c_1^{proj}, c_2^{proj}\} \\ &= \{\textit{(Flyover Traffic Decongestion),} \\ &\quad \textit{(Traffic Decongestion, E Traffic Decongestion)}\} \end{aligned}$$

Following the Galois Connection defined above, the abstract table $t_2^\sharp$ is depicted in Table VII. Consider a set of MDs $\Sigma = \{\psi_1, \psi_2\}$ where

$$\psi_1 = \{\mathtt{Manager}\} \rightarrow \{\mathtt{Project\ name}\}$$

$$\psi_2 = \{\mathtt{Project\ name}\} \rightarrow \{\mathtt{Location}\}$$

Applying MDs in the order ψ_1 ψ_2, we get the abstract clean instance in Table VIII. Observe that any further application of any of the MDs will not change the result, i.e. the result is least fixed point solution.

Case-2: Consider a set of MDs which forms a cycle, for instance $A_1 \rightarrow A_2$, $A_2 \rightarrow A_3$, $A_3 \rightarrow A_1$. In this case, we arrange the attributes in some order and identify the MD ψ with lowest attribute on the left hand side. We then apply MDs following case-1 starting with ψ. Observe that this may take exponential time to converge. A quick convergence in the abstract domain is possible by applying widening operation [9] which produces an over-approximated result.

Theorem 1 depicts that the abstraction function α preserves the similarity properties in the abstract domain as well.

Theorem 1. *Given two concrete values x_1 and x_2 such that $x_1 \approx x_2$ in concrete domain, where $\approx$ denotes similarity. The abstraction function α preserves similarity property, i.e. $\alpha(x_1) \approx^\sharp \alpha(x_2)$.*

Proof: Let $\alpha(x_1) = {L_1}^\sharp$ and $\alpha(x_2) = {L_2}^\sharp$. To prove, we first define the similarity metric $\approx^\sharp$ in the abstract domain. Let ${L_1}^\sharp$ and ${L_2}^\sharp$ be two sequences of bricks where ${L_1}^\sharp = [S_1]^{m_1,M_1}[S_2]^{m_2,M_2}$ and ${L_2}^\sharp = [S_3]^{n_1,N_1}[S_4]^{n_2,N_2}$.

$${L_1}^\sharp \approx^\sharp {L_2}^\sharp = \begin{cases} true & \text{if } must({L_1}^\sharp) = must({L_2}^\sharp) \\ false & \text{otherwise} \end{cases}$$

where the function $must(L^\sharp)$ returns the *must*-part of $L^\sharp$. Since α represents all similar strings in a cluster by an

Manager$^\sharp$	Project Name$^\sharp$	Location
$[Dileep]^{1,1}[Kumar]^{0,1}$	$[Flyover]^{1,1}[Traffic]^{1,1}[Decongestion]^{1,1}$	25 MG Road Patna
$[Dileep]^{1,1}[Kumar]^{0,1}$	$[E]^{0,1}[Traffic]^{1,1}[Decongestion]^{1,1}$	MG Road Patna
$[Aman]^{1,1}[Kumar]^{1,1}$	$[E]^{0,1}[Traffic]^{1,1}[Decongestion]^{1,1}$	Patna

TABLE VII: Abstract table $t_2{}^\sharp$

Manager$^\sharp$	Project Name$^\sharp$	Location$^\sharp$
$[Dileep]^{1,1}[Kumar]^{0,1}$	$[E, Flyover]^{0,1}[Traffic]^{1,1}[Decongestion]^{1,1}$	$[25]^{0,1}[MG]^{0,1}[Road]^{0,1}[Patna]^{1,1}$
$[Dileep]^{1,1}[Kumar]^{0,1}$	$[E, Flyover]^{0,1}[Traffic]^{1,1}[Decongestion]^{1,1}$	$[25]^{0,1}[MG]^{0,1}[Road]^{0,1}[Patna]^{1,1}$
$[Aman]^{1,1}[Kumar]^{1,1}$	$[E]^{0,1}[Traffic]^{1,1}[Decongestion]^{1,1}$	$[25]^{0,1}[MG]^{0,1}[Road]^{0,1}[Patna]^{1,1}$

TABLE VIII: Abstract clean instance applying in order ψ_1 ψ_2

Address$^\sharp$	Company$^\sharp$
$[Main]^{1,1}[St]^{1,1}[Ottawa]^{0,1}$	$[IBM]^{1,1}[India, International]^{0,1}$
$[25]^{0,1}[Main]^{1,1}[St]^{1,1}[Ottawa]^{0,1}$	$[IBM]^{1,1}[India, International]^{0,1}$
$[25]^{0,1}[Main]^{1,1}[St]^{1,1}$	$[IBM]^{1,1}[India, International]^{0,1}$

TABLE IX: Result of Q_1

abstract brick value, the *must*-part in the brick represents the common part of all the strings in the cluster (which includes both x_1 and x_2). This means, the *must*-part will be equal in both $L_1{}^\sharp$ and $L_2{}^\sharp$. Therefore, $L_1{}^\sharp \approx^\sharp L_2{}^\sharp$. ■

D. Defining Sound Query Answering

In order to obtain a sound query answering, we first apply abstraction to the query's variables/parameters so that we can compare it with the obtained abstract clean instance. Here also, we have to apply bricks as abstract domain (we have to apply the same abstract domain as we used before, to abstract the database).

Example 5. *Consider the abstract clean instance in Table IV. Consider the following query:*

$$Q_1 = \texttt{SELECT}\ \textbf{Address}, \textbf{Company}\ \texttt{FROM}\ t_1$$
$$\texttt{WHERE}\ \textbf{Name} = \text{"C Doe"}$$

The corresponding abstract version is:

$$Q_1^\sharp = \texttt{SELECT}\ \textbf{Address}^\sharp, \textbf{Company}^\sharp\ \texttt{FROM}\ t_1^\sharp$$
$$\texttt{WHERE}\ \textbf{Name}^\sharp = [C]^{1,1}[Doe]^{1,1}$$

The result[2] of $Q_1^\sharp$ on the clean instance $t_1^\sharp$ is depicted in Table IX. Observe that the result is an over-approximation of the concrete results. In fact, the result provides users more information than that in case of the concrete domain. This reduces the possibility to get "No Answer" in dirty databases. The computational complexity is less than [2], as there is no need to issue the same query on multiple clean instances and to combine the results. Moreover, users may identify the certain and possible answers from the result as well. For instance, the must-part of **Address**$^\sharp$ *in Table IX is* $[Main]^{1,1}[St]^{1,1}$ *whose concretization is* $\gamma([Main]^{1,1}[St]^{1,1})$=*{"Main St"}.*

[2]The detail sound query processing in the abstract domain is found in [17].

V. Complexity Analysis

Let n and m be the number of tuples and attributes respectively. The mapping of clusters into abstract domain is linear *w.r.t.* n and m. Given a set of r matching dependences, for each matching dependence, the tuple-comparison to identify the applicability of it requires $O(2^n)$ time. Our approach will improve the average case complexity, as the abstraction reduces a number of tuples into a single abstract tuple, reducing the comparison complexity. Space complexity in our approach is $O(1)$.

VI. Experimental Results

The experiment is performed on a system equipped with 2.50 GHz Intel $CORE^{TM}$ *i*3 processor, Windows 8.1 operating system, and 4GB memory. In order to compare our solution with previous approach, we have used the RESTAURANT dataset[3] which has 864 records from Fodor's and Zagat's restoumant guides containing 112 errors. For our experiment, we have considered three attributes `Name`, `Address` and `City`, and the MDs $\Sigma = \{\psi_1, \psi_2\}$ where

$$\psi_1 = \{\texttt{Name, Address}\} \rightarrow \{\texttt{City}\}$$

$$\psi_2 = \{\texttt{Name, Address, City}\} \rightarrow \{\texttt{Address}\}$$

The time in milliseconds for cleaning process is reported in tables X and XI respectively in case of both concrete (previous approach [2]) and abstract domains. These

Number of tuples	Time for Order-1 : $\psi_1 \rightarrow \psi_2$	Time for Order-2 : $\psi_2 \rightarrow \psi_1 \rightarrow \psi_2$	Total Time (millisecond)
50	305	360	665
100	340	380	720
150	450	495	945
250	515	631	1146
500	1679	2376	4055
864	4678	5223	9901

TABLE X: Cleaning result in concrete domain [2]

comparative results are shown graphically in Figure 2. Since in our case we have considered the time for database abstraction, this yields a little increase of the cleaning time. However, this static cleaning overhead is tolerable as it is done only once. In Table XII, we compare the number of clean instances generated in

[3]http://www.cs.utexas.edu/users/ml/riddle/data.html.

Number of tuples	Time for **Order1 :** $\psi_1 \rightarrow \psi_2$
50	400
100	800
150	1460
250	3895
500	8490
864	33675

TABLE XI: Cleaning result in abstract domain

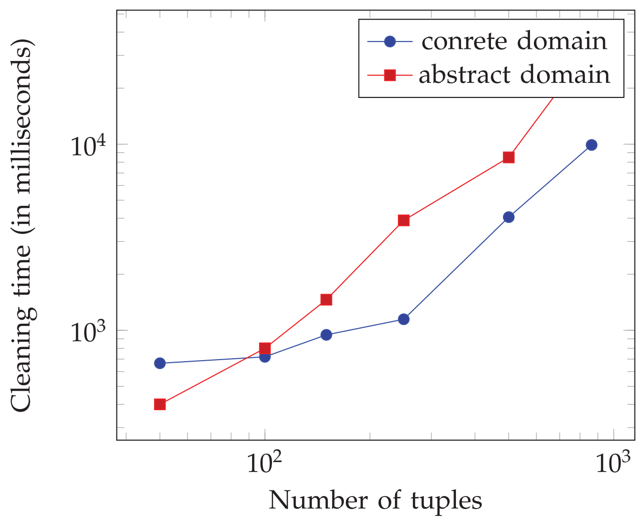


Fig. 2: Cleaning Time in Concrete and Abstract Domains

concrete and abstract domains respectively. This shows the most attractive feature of our approach: As in the abstract domain the cleaning process generates a single clean instance always, this reduces the query execution overhead significantly compared to the concrete one where query execution time is a multiplicative factor of the number of clean instances obtained.

Number of tuples	**Number of attributes**	**Number of MDs**	**Number of concrete clean instances**	**Number of abstract clean instances**
4	3	2	1	1
6	3	2	2	1
8	3	2	4	1
12	3	2	8	1
15	3	2	16	1

TABLE XII: Comparison of number of clean instances in concrete and abstract domain

VII. Conclusions

In this paper, we apply the Abstract Interpretation framework to clean dirty databases. The result is an over-approximation of all possible concrete clean instances. The approach significantly reduces the computational complexity and memory requirement. Our future aim is to integrate the notion of contexts during the cleaning process to improve the preciseness and efficiency of the query systems.

References

[1] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases: A probabilistic approach. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 30–30. IEEE, 2006.

[2] L. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. In *In Proc. ICDT*, 2011.

[3] G. Beskales, I. F. Ilyas, and L. Golab. Sampling the repairs of functional dependency violations under hard constraints. *Proceedings of the VLDB Endowment*, 3(1-2):197–207, 2010.

[4] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48. ACM, 2003.

[5] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 746–755. IEEE, 2007.

[6] L. Bravo, W. Fan, and S. Ma. Extending dependencies with conditions. In *Proceedings of the 33rd international conference on Very large data bases*, pages 243–254. VLDB Endowment, 2007.

[7] P. Christen. *Data Matching Concepts and Techniques for Record Linkage,Entity Resolution,and Duplicate detection*. Springer-Verlag, Research School of Computer Science ,The Australian National University,Canberra, ACT,Australia.

[8] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 458–469. IEEE, 2013.

[9] A. Cortesi and R. Halder. Abstract interpretation of recursive queries. In *Proc. of the 9th Int. Conf. on Distributed Computing and Internet Technologies*, pages 157–170, India, 5–8 Feb 2013 2013. Springer LNCS 7753.

[10] G. Costantini, P. Ferrara, and A. Cortesi. Static analysis of string values. In *Formal Methods and Software Engineering*, pages 505–521. Springer, 2011.

[11] P. Cousot. Abstract interpretation based formal methods and future challenges. In *Informatics*, pages 138–156. Springer, 2001.

[12] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. Nadeef: a commodity data cleaning system. In *Proceedings of the 2013 international conference on Management of data*, pages 541–552. ACM, 2013.

[13] A. Elgamal, N. Mosa, and N. Amasha. Application of framework for data cleaning to handle noisy data in data warehouse.

[14] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):1–16, 2007.

[15] W. Fan. Dependencies revisited for improving data quality. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 159–170. ACM, 2008.

[16] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. *Proceedings of the VLDB Endowment*, 2(1):407–418, 2009.

[17] R. Halder and A. Cortesi. Abstract interpretation of database query languages. *Computer Languages, Systems & Structures*, 38:123–157, 2012.

[18] M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Min. Knowl. Discov.*, 2(1):9–37, Jan. 1998.

[19] W. Kim, B.-J. Choi, E.-K. Hong, S.-K. Kim, and D. Lee. A taxonomy of dirty data. *Data mining and knowledge discovery*, 7(1):81–99, 2003.

[20] S. Kolahi and L. V. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *Proceedings of the 12th International Conference on Database Theory*, pages 53–62. ACM, 2009.

[21] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010.

[22] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. 2000.