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 in 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 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 that a 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,

- **Data integration errors**: It is 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 dependences 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 enforce 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 $\mathcal{D}^c$ and the abstract semantics domain $\mathcal{D}^a$ is formalized by Galois Connection $(\mathcal{D}^c, \alpha, \gamma, \mathcal{D}^a)$, where $\alpha$, $\gamma$ are the abstraction and concretization functions respectively. Let $\text{INT}$ be the set of integers and $\text{SIGN}$ be the abstract domain of sign. The Galois connection $(\varphi(\text{INT}), \alpha, \gamma, \text{SIGN})$ is represented below:

![Galois Connection Diagram]

When either $\alpha$ is surjective or $\gamma$ is injective, the Galois Connection $(\mathcal{D}^c, \alpha, \beta, \mathcal{D}^a)$ is called Galois Insertion. In this case, $\alpha \circ \gamma = \lambda \mathcal{D}^c. \mathcal{D}^a$.

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 \([\text{Name, Phone}] \rightarrow \text{Address}\)).

Matching function combines the information in those address values and generates \(t_0'\), assuming that the values of \text{Name} and \text{Phone} in \(t_0\) are similar.

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Doe</td>
<td>(613)1234</td>
<td>Main St, Ottawa</td>
</tr>
<tr>
<td>J Doe</td>
<td>12345</td>
<td>25 Main St, Ottawa</td>
</tr>
</tbody>
</table>

(a) Table \(t_0\)

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Doe</td>
<td>(613)1234</td>
<td>Main St, Ottawa</td>
</tr>
<tr>
<td>J Doe</td>
<td>12345</td>
<td>25 Main St, Ottawa</td>
</tr>
</tbody>
</table>

(b) Table \(t_0'\) after applying matching function on \text{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 \(X \rightarrow Y\) where \(X\) and \(Y\) 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 \(X\) are similar, then their \(Y\)-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 \(\text{attr}(t) = \{a_1, a_2, ..., a_n\}\). Let \(D_i\) be the domain of the attribute \(a_i\). The values in \(t\) corresponding to the attribute \(a_i\) is:

\[\text{Val}(a_i, t) = \pi_{a_i}(t) \subseteq D_i\]

where \(\pi\) represents projection operator. Applying similarity metric \(F\), we obtain a set of clusters

\[C(a_i, t) = F \circ \text{Val}(a_i, t) = \{c_1, c_2, ..., c_m\}\]

Example 1. To illustrate, let us consider the following table \(t_1\) in Table II. Let us assume that, applying a suitable clustering algorithm1, we obtain the following clusters on the attributes Name, Address and Company:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caezer Doe</td>
<td>Main St Ottawa</td>
<td>IBM India</td>
</tr>
<tr>
<td>C Doe</td>
<td>Main St</td>
<td>IBM</td>
</tr>
<tr>
<td>Christian</td>
<td>25 Main St</td>
<td>IBM International</td>
</tr>
<tr>
<td>Peter</td>
<td>25 MG Road Patna</td>
<td>Samsung Electronics</td>
</tr>
<tr>
<td>Christ</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

---

1Note 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 $C$ be the set of all clusters on a domain of values. We assume that the clusters may overlap each other. The powerset $\wp(C)$ forms a complete lattice $L_C = (\wp(C), \subseteq, C, \emptyset, \cup, \cap)$. Consider a complete lattice $L_A = (A, \subseteq, \top, \bot, \cup, \cap, \wedge)$ of an abstract values domain representing the properties of concrete values. We define the Galois Connection between $L_C$ and $L_A$ as $(L_C, C, \gamma, L_A)$. Given a concrete value $x$, we define the abstraction function as follows:

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

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 $\gamma$.

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

Example 2. Consider the concrete table $t_1$ in Table II. Following the Galois Connection defined above, we obtain the abstract version $t_1^\#$ 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:

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

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 $[S_1]_{M_1}$ and $[S_2]_{M_2}$, the matching function is defined as

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

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:

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

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"]".$^{1.1}$ 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 $X^\# \rightarrow Y^\#$ on $t^\#$ represents that if, for a set of tuples, the must-part of the abstract values corresponding to the attributes $X^\#$ are same, then their $Y^\#$ 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 $t_1^\#$ in Table III(b). Consider the MD: $\psi_1$ which states that if for two tuples the name, address and company are similar, then their companies must be matched (deonted $\psi_1 = \{\text{Name, Address, Company}\} \rightarrow \{\text{Company}\}$).

As for the first three tuples the must-part of the abstract values in attribute Name, Address, Company are similar, we apply $\psi_1$ 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,
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 in 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aman Kumar</td>
<td>Main St Ottawa</td>
<td>IBM India</td>
</tr>
<tr>
<td>Dileep</td>
<td>Main St</td>
<td>IBM India</td>
</tr>
<tr>
<td>Christian Doe</td>
<td>25 MG Road Patna</td>
<td>IBM International</td>
</tr>
<tr>
<td>Peter Christian</td>
<td>25 MG Road Patna</td>
<td>Samsung Electronics</td>
</tr>
<tr>
<td>Peter Christ</td>
<td>MG Road Patna</td>
<td>Samsung Electronics</td>
</tr>
</tbody>
</table>

TABLE VI: Table $t_2$

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

$$C(\text{Manager}, t_2) = \{c_1^{\text{Aman}}, c_2^{\text{Dileep}}\}$$

$$= \{\text{Dileep, Aman Kumar}\}, \text{ (Aman Kumar)}$$

$$C(\text{Project name}, t_2) = \{c_1^{\text{Dileep}}, c_2^{\text{Project}}\}$$

$$= \{\text{Traffic, Flyover, Traffic Deco}, \text{Traffic Deco}\}$$

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

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

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

Applying MDs in the order $\psi_1, \psi_2$, 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 $\psi$ with lowest attribute on the left hand side. We then apply MDs following case-1 starting with $\psi$. 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 $\alpha$ 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 $\alpha$ preserves similarity property, i.e. $\alpha(x_1) \approx \alpha(x_2)$.

Proof: Let $\alpha(x_1) = L_1^\alpha$ and $\alpha(x_2) = L_2^\alpha$. To prove, we first define the similarity metric $\approx^\alpha$ in the abstract domain. Let $L_1^\alpha$ and $L_2^\alpha$ be two sequences of bricks where $L_1^\alpha = [S_1]^{m_1} \otimes [S_2]^{m_2} \otimes [S_3]^{m_3}$ and $L_2^\alpha = [S_3]^{n_1} \otimes [S_4]^{n_2}$.

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

where the function $\text{must}(L^\alpha)$ returns the must-part of $L^\alpha$. Since $\alpha$ represents all similar strings in a cluster by an
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:

\[
Q_1 = \text{SELECT Address, Company FROM } t_1 \quad \text{WHERE Name = “C Doe”}
\]

The corresponding abstract version is:

\[
Q^A_1 = \text{SELECT Address}^A, \text{Company}^A \quad \text{FROM } t^A_1 \quad \text{WHERE Name}^A = [C]^1[Doe]^1
\]

The result \( Q^A_1 \) on the clean instance \( t^A_1 \) 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\(^A\) in Table IX is [Main]\(^A\)[St]\(^A\)[1],\(^1\) whose concretization is \( \gamma([\text{Main}]^1[\text{St}]^1) = “\text{Main St}” \).

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

TABLE XI: Cleaning result in abstract domain

<table>
<thead>
<tr>
<th>Number of tuples</th>
<th>Time for Order1 : ( \psi_1 \rightarrow \psi_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>400</td>
</tr>
<tr>
<td>100</td>
<td>800</td>
</tr>
<tr>
<td>150</td>
<td>1460</td>
</tr>
<tr>
<td>250</td>
<td>3895</td>
</tr>
<tr>
<td>500</td>
<td>8490</td>
</tr>
<tr>
<td>864</td>
<td>33675</td>
</tr>
</tbody>
</table>

Fig. 2: Cleaning Time in Concrete and Abstract Domains

clean 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.

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

<table>
<thead>
<tr>
<th>Number of tuples</th>
<th>Number of attributes</th>
<th>Number of MDs</th>
<th>Number of concrete clean instances</th>
<th>Number of abstract clean instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>

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