

Matching Dependencies with Arbitrary Attribute Values: Semantics, Query Answering and Integrity Constraints*

Jaffer Gardezi
University of Ottawa, SITE
Ottawa, Canada
jgard082@uottawa.ca

Leopoldo Bertossi[†]
Carleton University, SCS
Ottawa, Canada
bertossi@scs.carleton.ca

Iluju Kiringa
University of Ottawa, SITE
Ottawa, Canada
kiringa@site.uottawa.ca

ABSTRACT

Matching dependencies (MDs) were introduced to specify the identification or matching of certain attribute values in pairs of database tuples when some similarity conditions are satisfied. Their enforcement can be seen as a natural generalization of entity resolution. In what we call the *pure case* of MDs, any value from the underlying data domain can be used for the value in common that does the matching. We investigate the semantics and properties of data cleaning through the enforcement of matching dependencies for the pure case. We characterize the intended clean instances and also the *clean answers* to queries as those that are invariant under the cleaning process. The complexity of computing clean instances and clean answers to queries is investigated. Tractable and intractable cases depending on the MDs are identified.

1. INTRODUCTION

A database instance may contain several tuples and values in them that refer to the same external entity that is being modeled through the database. In consequence, the database may be modeling the same entity in different forms, as different entities, which most likely is not the intended representation. This problem could be caused by errors in data, by data coming from different sources that use different formats or semantics, etc. In this case, the database is considered to contain dirty data, and it must undergo a cleansing process that goes through two interlinked phases: detecting tuples (or values therein) that should be matched or identified, and, of course, doing the actual matching. This problem is usually called *entity resolution*, *data fusion*, *duplicate record detection*, etc. Cf. [13, 10] for some recent

*Research supported by the NSERC Strategic Network on Business Intelligence (BIN,ADC05) and NSERC/IBM CRDPJ/371084-2008.

[†]Faculty Fellow of the IBM CAS. Also affiliated to University of Concepción (Chile).

Permission to make digital or hard copies of all or part 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. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$10.00.

surveys and [6] for recent work in this area.

Quite recently, and generalizing entity resolution, [14, 15] introduced *matching dependencies* (MDs), which are declarative specifications of matchings of attribute values that should hold under certain conditions. MDs help identify duplicate data and enforce their merging by exploiting semantic knowledge expressed.

Loosely speaking, an MD is a rule defined on a database which states that, for any pair of tuples from given relations within the database, if the values of certain attributes of the tuples are similar, then the values of another set of attributes should be considered to represent the same object. In consequence, they should take the same values. Here, similarity of values can mean equality or a domain-dependent similarity relationship, e.g. related to some metric, such as the edit distance.

Example 1. Consider the following database instance of a relation P .

Name	Phone	Address
John Smith	723-9583	10-43 Oak St.
J. Smith	(750) 723-9583	43 Oak St. Ap. 10

Similarity of the names in the two tuples (as measured by, e.g. edit distance) is insufficient to establish that the tuples refer to the same person. This is because the last name is a common one, and only the first initial of one of the names is given. However, similarity of their phone and address values indicates that the two tuples may be duplicates. This is expressed by an MD which states that, if two tuples from P have similar address and phone, then the names should match. In the notation of MDs, this is expressed as

$$P[\text{Phone}] \approx P[\text{Phone}] \wedge P[\text{Address}] \approx P[\text{Address}] \rightarrow P[\text{Name}] \approx P[\text{Name}]. \quad \square$$

The identification in [14, 15] of a new class of dependencies and their declarative formulation have become important additions to data cleaning research. In this work we investigate matching dependencies, starting from and refining the model-theoretic and dynamic semantics of MDs introduced in [15].

Any method of querying a dirty data source must address the issue of duplicate detection in order to obtain accurate answers. Typically, this is done by first cleaning the data by discarding or combining duplicate tuples and standardizing formats. The result will be a new database where the entity conflicts have been resolved. However, the entity resolution problem may have different *solution instances* (which we will simply call *solutions*), i.e. different clean versions of the

original database. The model-theoretic semantics that we propose and investigate defines and characterizes the class of solutions, i.e. of intended clean instances.

After a clean instance has been obtained, it can be queried as usual. However, the query answers will then depend on the particular solution at hand. So, it becomes relevant to characterize those query answers that are invariant under the different (sensible) ways of cleaning the data, i.e. that persist across the solutions. This is an interesting problem *per se*. However, it becomes crucial if one wants to obtain semantically clean answers while still querying the original dirty data source.

This kind of virtual cleaning and query answering on top of it have been investigated in the area of *consistent query answering* (CQA) [3], where, instead of MDs, classical integrity constraints (ICs) are considered, and database instances are *repaired* in order to restore consistency (cf. [9, 7, 11] for surveys of CQA). Virtual approaches to robust query answering under entity resolution and enforcement of matching dependencies are certainly unavoidable in virtual data integration systems.

In this paper we make the following contributions, among others:

1. We revisit the semantics of MDs introduced in [15], pointing out sensible and justified modifications of it. A new semantics for MD satisfaction is then proposed and formally developed.
2. Using the new MD semantics, we formally define the intended solutions for a given, initial instance, D_0 , that may not satisfy a given set of MDs. They are called *minimally resolved instances* (MRIs) and are obtained through an iteration process that stepwise enforces the satisfaction of MDs until a stable instance is reached. The resulting instances minimally differ from D_0 in terms of number of changes of attribute values. This semantics (and the whole paper) considers the *pure case* introduced in [15], in the sense that the values than can be chosen to match attribute values are arbitrarily taken from the underlying data domains. No matching functions are considered, like in [6], for example (where entire tuples are merged, not individual attribute values).
3. We introduce the notion of *resolved answers* to a query posed to D_0 . They are the answers that are invariant under the MRIs.
4. We investigate the computability and complexity of computing MRIs and resolved answers, identifying cases where computing (actually, deciding) resolved answers is intractable.

This paper is organized as follows. Section 2 presents basic concepts and notations needed in the rest of the paper. Section 3 identifies some problems with the MD semantics, and refines it to address them. It also introduces the resolved instances and resolved answers to a query. Section 4 considers the problems of computing resolved instances and resolved query answers. Section 5 presents some final conclusions.

2. PRELIMINARIES

In general terms, we consider a relational schema \mathcal{S} that includes an enumerable infinite domain U . An instance D

of \mathcal{S} can be seen as a finite set of ground atoms of the form $R(\bar{t})$, where R is a database predicate in \mathcal{S} , and \bar{t} is a tuple of constants from U . We assume that each database tuple has an identifier, e.g. an extra attribute that acts as a key for the relation and is not subject to updates. In the following it will not be listed, unless necessary, as one of the attributes of a database predicate. It plays an auxiliary role only, to keep track of updates on the other attributes. $R(D)$ denotes the extension of R in D . We sometimes refer to attribute A of R by $R[A]$. If the i th attribute of predicate R is A , for a tuple $t = (c_1, \dots, c_j) \in R(D)$, $t[A]$ denotes the value c_i . The symbol $t[\bar{A}]$ denotes the vector whose entries are the values of the attributes in the vector \bar{A} . The attributes may have subdomains that are contained in U . Constants will be denoted by lower case letters at the beginning of the alphabet.

A matching dependency [14], involving predicates $R(A_1, \dots, A_n), S(B_1, \dots, B_m)$, is a rule of the form

$$\bigwedge_{i \in I, j \in J} R[A_i] \approx_{ij} S[B_j] \rightarrow \bigwedge_{i \in I', j \in J'} R[A_i] \equiv S[B_j]. \quad (1)$$

Here R and S could be the same predicate. I, I' and J, J' are fixed subsets of $\{1, \dots, n\}$ and $\{1, \dots, m\}$, resp. We assume that, when A_i, B_j are related via \approx_{ij} or \equiv in (1), they share the same (sub)domain, so their values can be compared by the domain-dependent binary similarity predicate, \approx_{ij} or can be identified, resp. In this paper, we will assume that there is at most one similarity operator defined on the domain of any given attribute.

The similarity operators, generically denoted with \approx , are assumed to have the properties of: (a) Symmetry: If $x \approx y$, then $y \approx x$. (b) Equality subsumption: If $x = y$, then $x \approx y$.

The MD in (1) is implicitly universally quantified in front and applied to pairs of tuples t_1, t_2 for R and S , resp. The expression $\bigwedge R[A_i] \approx_{ij} S[B_j]$ states that the values of the attributes A_i in tuple t_1 are similar to those of attributes B_j in tuple t_2 . If this holds, the expression $R[A_i] \equiv S[B_j]$ indicates that, for the same tuples t_1 and t_2 , $t_1[A_i]$ and $t_2[B_j]$ on the RHS should be updated so that they become the same, i.e. their values are identified or matched. However, the attribute values to be used for this matching are left unspecified by (1).

For abbreviation, we will sometimes write MDs as

$$R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \equiv S[\bar{E}], \quad (2)$$

where $\bar{A}, \bar{B}, \bar{C}$, and \bar{D} represent the lists of attributes, $(A_1, \dots, A_k), (B_1, \dots, B_k), (C_1, \dots, C_{k'}),$ and $(E_1, \dots, E_{k'})$, respectively. We refer to the pairs of attributes (A_i, B_i) and (C_i, E_i) as *corresponding pairs* of attributes of the pairs (\bar{A}, \bar{B}) and (\bar{C}, \bar{E}) , respectively. For an instance D and a pair of tuples $t_1 \in R(D)$ and $t_2 \in S(D)$, $t_1[\bar{A}] \approx t_2[\bar{B}]$ indicates that the similarities of the values for all corresponding pairs of attributes of (\bar{A}, \bar{B}) hold. Similarly, $t_1[\bar{C}] = t_2[\bar{E}]$ denotes the equality of the values of all pairs of corresponding attributes of (\bar{C}, \bar{E}) .

Since an MD involves an update operation, the MD is a condition that is satisfied by a pair of database instances: an instance D and its updated instance D' .

Definition 1. [15] Let D, D' be instances of schema \mathcal{S} with predicates R and S , such that, for each tuple t in D , there is a unique tuple t' in D' with the same identifier as t , and viceversa. The pair (D, D') satisfies the MD m in (2), denoted $(D, D') \models_F m$, iff, for every pair of tuples $t_R \in R(D)$

and $t_S \in S(D)$, if t_R and t_S satisfy $t_R[\bar{A}] \approx t_S[\bar{B}]$, then for the corresponding tuples t'_R and t'_S in $R(D')$, $S(D')$, resp., it holds: (a) $t'_R[\bar{C}] = t'_S[\bar{E}]$, and (b) $t'_R[\bar{A}] \approx t'_S[\bar{B}]$. \square

Intuitively, D' in Definition 1 is an instance obtained from D by enforcing m on instance D . For a set M of MDs, and a pair of instances (D, D') , $(D, D') \models_F M$ means that $(D, D') \models_F m$, for every $m \in M$.

An instance D' is *stable* [15] for a set M of MDs if $(D', D') \models_F M$. Stable instances correspond to the intuitive notion of a clean database, in the sense that all the expected value identifications already take place in it. Although not explicitly developed in [15], for an instance D , if $(D, D') \models_F M$ for a stable instance D' , then D' is expected to be reached as a fix-point of an iteration of value identification updates that starts from D and is based on M .

3. MD SEMANTICS REVISITED

Condition (b) in Definition 1 is used to avoid that the identification updates destroy the original similarities. Unfortunately, enforcing the requirement sometimes leads to counterintuitive results.

Example 2. Consider the following instance D with string-valued attributes, and MDs:

R	A	B	C	S	E	F
	a	c	g		h	c
	a	c	ksp		mSP	c

$$R[A] \approx R[A] \rightarrow R[C] \neq R[C] \quad (3)$$

$$R[C] \approx S[E] \rightarrow R[B] \neq S[F] \quad (4)$$

For two strings s_1 and s_2 , $s_1 \approx s_2$ if the edit distance d between s_1 and s_2 satisfies $d \leq 1$. To produce an instance D' satisfying $(D, D') \models_F M$, the strings g and ksp must be changed to some common string s' .

Because of the similarities $h \approx g$ and $ksp \approx mSP$, s' must be similar to the E attribute values of the tuples in S , by condition (b) of Definition 1 and MD (4). Clearly, there is no s' that is similar to both h and mSP . Therefore, at least one of h and mSP must be modified to some new value in D' . \square

Another problem with the semantics of MDs is that it allows duplicate resolution in instances that are already resolved. Intuitively, there is no reason to change the values in an instance that is stable for a set of MDs M , because there is no reason to believe, on the basis of M , that these values are in error. However, even if an instance D satisfies $(D, D) \models_F M$, it is always possible, by choosing different common values, to produce a different instance D' such that $(D, D') \models_F M$. This is illustrated in the next example.

Example 3. Let D be the instance below and the MD $R[A] \approx R[A] \rightarrow R[B] \neq R[B]$.

R	A	B
	a	c
	a	c

Although D is stable, $(D, D') \models_F m$ is true for any D' where the B attribute values of the two tuples are the same. \square

3.1 MD satisfaction

We now propose a new semantics for MD satisfaction that disallows unjustified attribute modifications. We keep condition (a) of Definition 1, while replacing condition (b) with a restriction on the possible updates that can be made.

Definition 2. Let D be an instance of schema \mathcal{S} , $R \in \mathcal{S}$, $t_R \in R(D)$, C an attribute of R , and M a set of MDs. Value $t_R[C]$ is *modifiable* if there exist $S \in \mathcal{S}$, $t_S \in S(D)$, an $m \in M$ of the form $R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \neq S[\bar{E}]$, and a corresponding pair (C, E) of (\bar{C}, \bar{E}) , such that one of the following holds: 1. $t_R[\bar{A}] \approx t_S[\bar{B}]$, but $t_R[C] \neq t_S[E]$. 2. $t_R[\bar{A}] \approx t_S[\bar{B}]$ and $t_S[E]$ is modifiable. \square

Example 4. Consider two relations R and S with two MDs defined on them:

R	A	B	S	C	E
t_0	a_0	b	t_3	a_3	c
t_1	a_1	b	t_4	a_4	c
t_2	a_2	b	t_5	a_5	c

$$m_1 : R[A] \approx R[A] \rightarrow R[B] \neq R[B],$$

$$m_2 : R[A] \approx S[C] \rightarrow R[B] \neq S[E],$$

$$m_3 : S[C] \approx S[C] \rightarrow S[E] \neq S[E].$$

The following similarities hold on the distinct constants of R and S : $a_i \approx a_{(i+1) \bmod 6}$, $0 \leq i \leq 5$. The values $t_2[B]$ and $t_3[E]$ are modifiable by condition 1 of Definition 2, m_2 , $a_2 \approx a_3$, and $t_2[B] \neq t_3[E]$. For the same reason, $t_0[B]$ and $t_5[E]$ are modifiable.

Value $t_1[B]$ is modifiable by condition 2 of Definition 2, m_1 , $a_1 \approx a_2$, and the fact that $t_2[B]$ is modifiable. Similarly, $t_4[E]$ is modifiable. \square

Definition 3. Let D, D' be instances for \mathcal{S} with the same tuple ids, and M a set of MDs. (D, D') satisfies M , denoted $(D, D') \models M$, iff:

- For any pair of tuples $t_R \in R(D)$, $t_S \in S(D)$, if there exists an MD in M of the form $R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \neq S[\bar{E}]$ and $t_R[\bar{A}] \approx t_S[\bar{B}]$, then for the corresponding tuples $t'_R \in R(D')$ and $t'_S \in S(D')$, it holds $t'_R[\bar{C}] = t'_S[\bar{E}]$.
- For any tuple $t_R \in R(D)$ and any attribute G of R , if $t_R[G]$ is not modifiable, then $t'_R[G] = t_R[G]$. \square

Condition 2. captures a natural default condition of persistence of values: those that have to be changed are changed only. As before, we define *stable* instance for M to mean $(D, D) \models M$. Except where otherwise noted, these are the notions of satisfaction and stability that we will use in the rest of this paper.

Example 5. Consider again example 4. The set of all D' such that $(D, D') \models M$ is the set of all instances obtained from D by changing all values of $R[B]$ and $S[E]$ to a common value, and leaving all other values unchanged. This is because the values of $R[B]$ and $S[E]$ are the only modifiable values, and these values must be equal by condition 1 of Definition 3 and the given similarities. \square

Condition 2 in Definition 3 on the set of updatable values does not prevent us from obtaining instances D' that enforce the MD, as the following theorem establishes.

Theorem 1. For any instance D and set of MDs M , there exists a D' such that $(D, D') \models M$. Moreover, for any attribute value that is changed from D to D' , the new value can be chosen arbitrarily, as long as it is consistent with $(D, D') \models M$. \square

The new semantics introduced in Definition 3 solves the problems mentioned at the beginning of this section. Notice that it does not require additional changes to preserve similarities (if the original ones were broken). Furthermore, modifications of instances, unless required by the enforcement of matchings as specified by the MDs, are not allowed. Also notice that the instance D' in Theorem 1 is not guaranteed to be stable. We address this issue in the next section.

Moreover, as can be seen from the proof of Theorem 1, the new restriction imposed by Definition 3 is as strong as possible in the following sense: Any definition of MD satisfaction that includes condition 1. must allow the modification of the modifiable attributes (according to Definition 2). Otherwise, it is not possible to ensure, for arbitrary D , the existence of an instance D' with $(D, D') \models M$.

3.2 Resolved instances

According to the MD semantics in [15], although not explicitly stated there, a clean version D' of an instance D is an instance D' satisfying the conditions $(D, D') \models M$ and $(D', D') \models M$. Due to the natural restrictions on updates captured by the new semantics (cf. Definition 3), the existence of such a D' is not guaranteed. Essentially, this is because D' is the result of a series of updates. The MDs are applied to the original instance D to produce a new instance, which may have new pairs of similar values, forcing another application of the MDs, which in their turn produces another instance, and so on, until a stable instance D' is reached. The pair (D, D') may not satisfy M . However, we will be interested in those instances D' just mentioned. The idea is to relax the condition $(D, D') \models M$, and obtain a stable D' after an iterative process of MD enforcement, which at each step, say k , makes sure that $(D_{k-1}, D_k) \models M$.

Definition 4. Let D be a database instance and M a set of MDs. A *resolved instance* for D wrt M is an instance D' , such that there is a finite (possibly empty) sequence of instances D_1, D_2, \dots, D_n with: $(D, D_1) \models M$, $(D_1, D_2) \models M, \dots$ $(D_{n-1}, D_n) \models M$, $(D_n, D') \models M$, and $(D', D') \models M$. \square

Note that, by Definition 3, for an instance D satisfying $(D, D) \models M$, it holds $(D, D') \models M$ if and only if $D' = D$. In this case, the only possible set of intermediate instances is the empty set and D is the only resolved instance. Thus, a resolved instance cannot be obtained by making changes to an instance that is already resolved.

Theorem 2. Given an instance D and a set M of MDs, there always exists a resolved instance of D with respect to M . \square

Example 6. Consider the following instance D of a relation R and set M of MDs:

$R(D)$	A	B	C
	a	b	d
	a	c	e
	a	b	e

$$R[A] \approx R[A] \rightarrow R[B] \approx R[B],$$

$$R[B] \approx R[B] \rightarrow R[C] \approx R[C].$$

All pairs of distinct constants in R are dissimilar. Two resolved instances D_1 and D_2 of R are shown.

$R(D_1)$	A	B	C
	a	b	d
	a	b	d
	a	b	d

$R(D_2)$	A	B	C
	a	b	e
	a	b	e
	a	b	e

Notice that $(D, D_1) \not\models M$, because the value of the C attribute of the second tuple is not modifiable in D . \square

The notion of resolved instance is one step towards the characterization of the intended clean instances. However, it still leaves room for refinement. Actually, the resolved instances that are of most interest for us are those that are somehow closest to the original instance. This consideration leads to the concept of *minimal resolved instance*, which uses as a measure of change the number of values that were modified to obtain the clean database. In Example 6, instance D_2 is a minimal resolved instance, whereas D_1 is not.

Definition 5. Let D be an instance.

- (a) $T_D := \{(t, A) \mid t \text{ is the id of a tuple in } D \text{ and } A \text{ is an attribute of the tuple}\}$.
- (b) $f_D : T_D \rightarrow U$ is given by: $f_D(t, A) :=$ the value for A in the tuple in D with id t .
- (c) For an instance D' with the same tuple ids as D :
 $S_{D,D'} := \{(t, A) \in T_D \mid f_D(t, A) \neq f_{D'}(t, A)\}$. \square

Intuitively, $S_{D,D'}$ is the set of all values changed in going from D to D' .

Definition 6. Let D be an instance and M a set of MDs. A *minimally resolved instance* (MRI) of D wrt M is a resolved instance D' such that $|S_{D,D'}|$ is minimum, i.e. there is no resolved instance D'' with $|S_{D,D''}| < |S_{D,D'}|$. We denote by $Res(D, M)$ the set of minimal resolved instances of D wrt the set M of MDs. \square

Example 7. Consider the instance below and the MD $R[A] \approx S[C] \rightarrow R[B] \approx S[D]$.

R	A	B
	a_1	b_1

S	C	D
	c_1	d_1

Assuming that $a_1 \approx c_1$, this instance has two minimal resolved instances, namely

R	A	B
	a_1	d_1

S	C	D
	c_1	d_1

R	A	B
	a_1	b_1

S	C	D
	c_1	b_1

 \square

Considering that MDs concentrate on changes of attribute values, we consider that this notion of minimality is appropriate. The comparisons have to be made at the attribute value level. Notice that in CQA a few other notions of minimality and comparison of instances have been investigated [7].

The requirement of definition 6 that the number changes be minimized in an MRI can be relaxed to allow MRIs whose change is within some percentage of the minimum. All the

results of this paper go through with minor modifications. This might be a more appropriate definition in certain duplicate resolution settings.

In this subsection, we defined the MRIs, which we use as our model of a clean database instance. This leads to the definition of resolved answers to a query in the next subsection. Intuitively, these are the answers that are true in all MRIs. This is analogous to consistent query answering, in which a set of consistent answers to a query is defined that is true in all minimal repairs of a database that violates a set of integrity constraints [3]. Indeed, instead of applying the dynamic semantics of MDs to this context, we could have taken a more traditional approach in which the MDs are interpreted as integrity constraints and the consistent answers are computed relative to these constraints. However, such an approach would not fully capture the semantic information contained in the MDs, as the next example shows.

Example 8. Consider the following instance of a relation R and MDs:

R	A	B	C
	a	b	c
	a	d	e

$$R[A] = R[A] \rightarrow R[B] \rightleftharpoons R[B]$$

$$R[B] = R[B] \rightarrow R[C] \rightleftharpoons R[C]$$

where all pairs of distinct constants in R are unequal. Suppose we viewed the MDs as functional dependencies to be satisfied by R , replacing \rightleftharpoons with $=$. Then the only inconsistent values are those in the B column. In the context of duplicate resolution, the appropriate way to repair R would be to set these values to a common value. However, this would result in an instance which violates the second constraint. In failing to consider the effect of updates, an approach that defines MDs as traditional integrity constraints will not take all duplicates into account and will not provide an appropriate semantics for the certain answers. It is easily verified that this approach is equivalent to ours in the case of the non-interacting MDs defined in section 4. \square

3.3 Resolved answers

Let $\mathcal{Q}(\bar{x})$ be a query expressed in the first-order language $L(\mathcal{S})$ associated to schema \mathcal{S} . Now we are in position to characterize the admissible answers to \mathcal{Q} from D , as those that are invariant under the matching resolution process.

Definition 7. A tuple of constants \bar{a} is a *resolved answer* to $\mathcal{Q}(\bar{x})$ wrt the set M of MDs, denoted $D \models_M \mathcal{Q}[\bar{a}]$, iff $D' \models \mathcal{Q}[\bar{a}]$, for every $D' \in \text{Res}(D, M)$. We denote with $\text{ResAn}(D, \mathcal{Q}, M)$ the set of resolved answers to \mathcal{Q} from D wrt M . \square

Example 9. (example 7 continued) The set of resolved answers to the query $\mathcal{Q}_1(x, y) : R(x, y)$ is empty since there are no tuples that are in the instance of R in all minimal resolved instances. On the other hand, the set of resolved answers to the query $\mathcal{Q}_2(x) : \exists y(R(x, y) \wedge (y = b_1 \vee y = d_1))$ is $\{a_1\}$. \square

In Section 4 we will study the complexity of the problem of computing the resolved answers, which we now formally introduce.

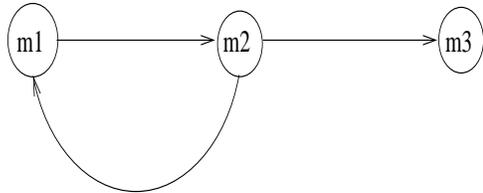


Figure 1: An MD-Graph

Definition 8. Given a schema \mathcal{S} , a query $\mathcal{Q}(\bar{x}) \in L(\mathcal{S})$, and a set M of MDs, the *Resolved Answer Problem* (RAP) is the problem of deciding membership of the set

$$RA_{\mathcal{Q}, M} := \{(D, \bar{a}) \mid \bar{a} \text{ is a resolved answer to } \mathcal{Q} \text{ from instance } D \text{ wrt } M\}.$$

If \mathcal{Q} is a boolean query, it is the problem of determining whether \mathcal{Q} is true in all minimal resolved instances of D . \square

4. COMPUTING RESOLVED INSTANCES AND ANSWERS

In this section, we consider the complexity of the $RA_{\mathcal{Q}, M}$ problem introduced in the previous section. For this goal it is useful to associate a graph to the set of MDs. We need a few notions before introducing it.

Definition 9. A set M of MDs is in *standard form* if no two MDs in M have the same expression to the left of the arrow. \square

Notice that any set of MDs can be put in standard form by replacing subsets of MDs of the form $\{R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}_1] \rightleftharpoons S[\bar{E}_1], \dots, R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}_n] \rightleftharpoons S[\bar{E}_n]\}$ by the single MD $R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \rightleftharpoons S[\bar{E}]$, where the set of corresponding pairs of attributes of (\bar{C}, \bar{E}) is the union of those of $(\bar{C}_1, \bar{E}_1), \dots, (\bar{C}_n, \bar{E}_n)$. From now on, we will assume that all sets of MDs are in standard form.

For an MD m , $\text{LHS}(m)$ and $\text{RHS}(m)$ denote the sets of attributes that appear to the left side and to right side of the arrow, respectively.

Definition 10. Let M be a set of MDs in standard form. The *MD-graph* of M , denoted $MDG(M)$, is a directed graph with a vertex labeled m for each $m \in M$, and with an edge from m_1 to m_2 iff $\text{RHS}(m_1) \cap \text{LHS}(m_2) \neq \emptyset$. \square

Example 10. Consider the set of MDs: $m_1 : R[A] \approx S[B] \rightarrow R[C] \rightleftharpoons S[D]$. $m_2 : R[C] \approx S[D] \rightarrow R[A] \rightleftharpoons S[B]$. $m_3 : S[E] \approx S[B] \rightarrow T[F] \rightleftharpoons T[F]$. It has the MD-graph shown in Figure 1. \square

A set of MDs whose MD-graph contains edges is called *interacting*. Otherwise, it is *non-interacting*.

We will use the following notions to discuss the tractability of computing resolved answers.

Definition 11. Let D be a database instance and M a set of MDs. A triple (t, A, v) , with $(t, A) \in T_D$ is a *certain triple* if $f_{D'}(t, A) = v$ in all MRIs D' of D . \square

Definition 12. A set of MDs M defined on a database schema \mathcal{S} is *hard* if it is an NP-hard problem to determine, given an instance D with schema \mathcal{S} and set of certain triples P , whether P is the set of all certain triples. The set M is *easy* if this can be determined in polynomial time. \square

RAP is intractable for hard sets of MDs even for some very simple queries. The following result is straightforward.

Definition 13. For a query Q and set of MDs M defined on a schema S , $RAS_{Q,M}$ is the problem of determining, given an instance D with schema S and a set P of answers to Q , whether P is the set of all resolved answers to Q on D . \square

Theorem 3. Let M be a hard set of MDs defined on a schema S . Then there exists a single atom conjunctive query Q on S for which $RAS_{Q,M}$ is NP-hard. \square

It is straightforward that all sets of non-interacting MDs are easy. We now turn to the simplest case of interacting MDs: a set M of two MDs such that $MDG(M)$ has a single directed edge from one vertex to the other. For the complexity results of this section, we make the assumption for all similarity operators that there exists an infinite set of pairwise dissimilar elements.

Definition 14. An ordered pair (m_1, m_2) of MDs is a linear pair of MDs if the MD graph of $\{m_1, m_2\}$ has a single edge, and this edge is from m_1 to m_2 . \square

For a pair of database instances D and D' , $(D, D') \models M$ implies that certain groups of values in D must be set to a common value in D' . Since all similarity operators subsume equality, these values are similar in D' . We call similarities of D' which hold in D or which are implied by $(D, D') \models M$ *intended similarities*. Other new similarities can also arise in the updated instance D' , which we call *accidental similarities*. These similarities result from the particular choice of update value, and do not occur in all D' satisfying $(D, D') \models M$.

Example 11. Consider the two-attribute relation R given below, and the MD $R[A] = R[A] \rightarrow R[B] \Leftarrow R[B]$.

	A	B
t_1 :	a	c
t_2 :	a	e
t_3 :	b	d
t_4 :	b	f

One possible updated instance is

	A	B
t_1 :	a	d
t_2 :	a	d
t_3 :	b	d
t_4 :	b	d

Among the B attribute values, the intended similarities are $t_1[B] = t_2[B]$ and $t_3[B] = t_4[B]$, and the accidental similarities are $t_1[B] = t_3[B]$, $t_1[B] = t_4[B]$, $t_2[B] = t_3[B]$, and $t_2[B] = t_4[B]$. \square

Accidental similarities are a source of intractability for the computation of resolved answers when there are interacting MDs. This is because accidental similarities produced by the application of one MD affect the application of other MDs, leading to a dependence on the choices of common values.

Theorem 4. The following set of MDs is hard:

$$\begin{aligned} R[A] \approx R[A] \rightarrow R[B] \Leftarrow R[B] \\ R[B] \approx R[B] \rightarrow R[C] \Leftarrow R[C] \end{aligned}$$

\square

Linear pairs of MDs nonetheless can sometimes be easy in spite of the occurrence of accidental similarities. This happens when the interaction between the MDs is more restricted in the sense that accidental similarities generated by one MD cannot affect the application of the other MD. For example, the set of MDs

$$\begin{aligned} R[A] \approx R[A] \rightarrow R[B] \Leftarrow R[B] \\ R[A] \approx R[A] \wedge R[B] \approx R[B] \rightarrow R[C] \Leftarrow R[C] \end{aligned}$$

is easy. Intuitively, the conjunct $R[A] \approx R[A]$ in the second MD “filters out” the accidental similarities among the values of attributes in the B column, allowing only the intended similarities to be passed on to the C column by the second MD. More generally, the following can be proved.

Theorem 5. Any pair (m_1, m_2) of linear MDs of the form

$$\begin{aligned} m_1 : R[\bar{A}] \approx_1 S[\bar{B}] \rightarrow R[\bar{C}] \Leftarrow S[\bar{E}] \\ m_2 : R[\bar{A}] \approx_1 S[\bar{B}] \wedge R[\bar{F}] \approx_2 S[\bar{G}] \rightarrow R[\bar{H}] \Leftarrow S[\bar{I}] \end{aligned}$$

is easy. \square

5. CONCLUSIONS

In this paper we have proposed a revised semantics for matching dependency (MD) satisfaction wrt the one originally proposed in [15]. The main outcomes from that semantics are the notions of *minimally resolved instance* (MRI) and *resolved answers* (RAs) to queries. The former capture the intended, clean instances obtained after enforcing the MDs on a given instance. The latter are query answers that persist across all the MRIs, and can be considered as robust and semantically correct answers.

We investigated the new semantics, the MRIs and the RAs. We considered the existence of MRIs and derived some preliminary results on the complexity of computing the RAs. In future work, we will derive syntactic criteria on MDs for identifying what we called easy and hard sets of MDs. We will also investigate query rewriting methods for obtaining the RAs in the easy cases.

In this paper we have not considered cases where the matchings of attribute values, whenever prescribed by the MDs’ conditions, are made according to matching functions. This element adds an entirely new dimension to the semantics and the problems investigated here. It certainly deserves investigation.

6. REFERENCES

- [1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, Don Mills, Ontario, 1995.
- [2] F. Afrati and P. Kolaitis. Repair checking in inconsistent databases: Algorithms and complexity. In *Proc. ICDT*, pages 31–41, 2009.
- [3] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. PODS*, pages 68–79, 1999.
- [4] M. Arenas, L. Bertossi, and J. Chomicki. Answer sets for consistent query answering in inconsistent databases. *Theory and Practice of Logic Programming*, 3(4-5):393–424, 2003.
- [5] P. Barceló, L. Bertossi, and L. Bravo. Characterizing and computing semantically correct answers from databases with annotated logic and answer sets. In *Semantics in Databases*, pages 7–33, 2003.

- [6] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. Euijong Whang, and J. Widom. Swoosh: A generic approach to entity resolution. *VLDB Journal*, 18(1):255–276, 2009.
- [7] L. Bertossi. Consistent query answering in databases. *ACM Sigmod Record*, 35(2):68–76, 2006.
- [8] L. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Information Systems*, 33(4):407–434, 2008.
- [9] L. Bertossi and J. Chomicki. Query answering in inconsistent databases. In *Logics for Emerging Applications of Databases*, pages 43–83. Springer, 2003.
- [10] J. Bleiholder and F. Naumann. Data fusion. *ACM Computing Surveys*, 41(1):1–41, 2008.
- [11] J. Chomicki. Consistent query answering: Five easy pieces. In *Proc. ICDT*, pages 1–17, 2007.
- [12] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1/2):90–121, 2005.
- [13] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.
- [14] J. Fan. Dependencies revisited for improving data quality. In *Proc. PODS*, pages 159–170, 2008.
- [15] J. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. In *Proc. VLDB*, pages 407–418, 2009.
- [16] S. Flesca, F. Furfaro, and F. Parisi. Consistent query answers on numerical databases under aggregate constraints. In *Proc. DBPL*, pages 279–294, 2005.
- [17] E. Franconi, A. Laureti Palma, N. Leone, S. Perri, and F. Scarcello. Census data repair: A challenging application of disjunctive logic programming. In *Proc. LPAR*, pages 561–578, 2001.
- [18] A. Fuxman and R. Miller. First-order query rewriting for inconsistent databases. *J. Computer and System Sciences*, 73(4):610–635, 2007.
- [19] G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1389–1408, 2003.
- [20] A. Lopatenko and L. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *Proc. ICDT*, pages 179–193, 2007.
- [21] J. Wijsen. Database repairing using updates. *ACM Transactions on Database Systems*, 30(3):722–768, 2005.
- [22] J. Wijsen. Consistent query answering under primary keys: A characterization of tractable cases. In *Proc. ICDT*, pages 42–52, 2009.
- [23] J. Wijsen. On the consistent rewriting of conjunctive queries under primary key constraints. *Information Systems*, 34(7):578–601, 2009.
- [24] J. Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *Proc. PODS*, pages 179–190, 2010.

A. AUXILIARY RESULTS AND PROOFS

Proof of Theorem 1: Consider an undirected graph G whose vertices are labelled by pairs (t, A) , where t is a tuple identifier and A is an attribute of t . There is an edge between two vertices (s, A) and (t, B) iff s and t satisfy the similarity condition of some MD $m \in M$ such that A and B are matched by m .

Update D as follows. Choose a vertex (t_1, A) such that there is another vertex (t_2, B) connected to (t_1, A) by an edge and $t_1[A]$ and $t_2[B]$ must be made equal to satisfy the equalities in condition 1. of Definition 3. For convenience in this proof, we say that t_2 is unequal to t_1 for such a pair of tuples t_1 and t_2 . Perform a breadth first search (BFS) on G starting with (t_1, A) as level 0. During the search, if a tuple is discovered at level $i + 1$ that is unequal to an adjacent tuple at level i , the value of the attribute in the former tuple is modified so that it matches that of the latter tuple. When the BFS has completed, another vertex with an adjacent unequal tuple is chosen and another BFS is performed. This continues until no such vertices remain. It is clear that the resulting updated instance D' satisfies condition 1. of definition 3.

We now show by induction on the levels of the breadth first searches that for all vertices (t, A) visited, $t[A]$ is modifiable. This is true in the base case, by choice of the starting vertex. Suppose it is true for all levels up to and including the i^{th} level. By definition of the graph G and condition 2. of definition 2, the statement is true for all vertices at the $(i + 1)^{th}$ level. This proves the first statement of the theorem.

To prove the second statement, we show that, to satisfy condition 1. of Definition 3, the attribute values represented by each vertex in each connected component of G must be changed to a common value in the new instance. The statement then follows from the fact that the update algorithm can be modified so that the attribute value for the initial vertex in each BFS is updated to some arbitrary value at the start (since it is modifiable). By condition 1. of Definition 3, the pairs of values that must be equal in the updated instance D' correspond to those vertices that are connected by an edge in G . This fact and transitivity of equality imply that all attribute values in a connected component must be updated to a common value. \square

Proof of Theorem 2: We give an algorithm to compute a resolved instance, and use a monotonicity property to show that it always terminates. For attribute domain d in D , consider the set S^d of pairs (t, A) such that attribute A of the tuple with identifier t has domain d . Let $\{S_1, S_2, \dots, S_n\}$ be a partition of S^d into sets such that all tuple/attribute pairs in a set have the same value in D . Define the level of (t, A) to mean $|S_j|$ where $(t, A) \in S_j$.

The algorithm first applies all MDs in M to D by setting equal pairs of unequal values according to the MDs. Specifically, consider a connected component C of the graph in the proof of Theorem 1. If the values of $t[A]$ for all pairs (t, A) in C are not all the same, then their values are modified to a common value which is that of the pair with the highest level. This update is allowed by Theorem 1. In the case of a tie, the common value is chosen as the largest of the values according to some total ordering of the values from the domain that occur in the instance. It is easily verified

that this operation increases the sum over all the levels of the elements of S^d , where d is the domain of the attributes of the pairs in C . These updates produce an instance D_1 such that $(D, D_1) \models M$.

The MDs of M are then applied to the instance D_1 to obtain a new instance D_2 such that $(D_1, D_2) \models M$ and so on, until a stable instance is reached. For each new instance, the sum over all domains d of the levels of the $(t, A) \in S^d$ is greater than for the previous instance. Since this quantity is bounded above, the algorithm terminates with a resolved instance. \square

Proof of Theorem 4: The proof is by reduction from MONOTONE 3-SAT. Given an instance F of MONOTONE 3-SAT with clauses c_1, c_2, \dots, c_n , construct an instance of R with tuples t_1, t_2, \dots, t_{3n} . The sets $\{t_1, t_2, t_3\}, \{t_4, t_5, t_6\}, \dots$ are called 3-blocks. For $0 \leq i < n$, $t_{3i+1}[A] = t_{3i+2}[A] = t_{3i+3}[A] = k_i$ and for $0 \leq i < n$, the k_i are pairwise dissimilar. We refer to a clause as a positive (negative) clause if it contains only positive (negative) literals. If c_i is a positive clause, then $t_{3(i-1)+1}[C] = t_{3(i-1)+2}[C] = t_{3(i-1)+3}[C] = a$, and if c_i is a negative clause (contains only negative literals), then $t_{3(i-1)+1}[C] = t_{3(i-1)+2}[C] = t_{3(i-1)+3}[C] = b$. The values in the B column consist of a set S of pairwise dissimilar values, one for each variable in F . The values of $t_{3(i-1)+1}[B]$, $t_{3(i-1)+2}[B]$, and $t_{3(i-1)+3}[B]$ are the values in S corresponding to the variables in c_i .

In a resolved instance, the B attribute values of all tuples in a 3-block must be equal (because of the first MD). Minimal change in the B column is achieved by choosing as the common value any of the original B attribute values in the 3-block. We will show that there is a resolved instance with this choice of values for the B column and with no change to the values in the C column iff F is satisfiable. This is the only MRI when F is satisfiable. Thus, it is NP-hard to determine which values can change in an MRI, implying that the set of MDs is hard.

In a satisfying assignment to F , there is a literal for each clause in F that is made true by the assignment. For each 3-block in F , choose as the common B attribute value the value corresponding to the true literal for a satisfying assignment. Since the assignment is consistent, the values chosen for the 3-blocks corresponding to positive clauses are dissimilar to those corresponding to negative clauses. This implies that the second (and final) update will set to a common value exactly those sets of values in the C column that had a common value in the original instance. Choosing the original values as the update values, there is no change in the C column.

Conversely, suppose there is no satisfying assignment to F . Then, if a variable is chosen from each clause, there must be a negative and positive clause such that the same variable was chosen from them (otherwise F could be made true by setting the variables chosen from positive clauses true and setting those chosen from negative clauses false). This

implies that, when update values are chosen so as to achieve minimal change in the B column in the first update, the second update must set to a common value sets of values in the C column that were originally distinct. Specifically, pairs of 3-blocks whose C attribute values were originally distinct must have their C attribute values set to a common value. \square

For the next proof, we need an auxiliary definition and result.

Definition 15. Let m be the MD $R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \Leftarrow S[\bar{E}]$. The transitive closure, T^\approx , of \approx is the transitive closure of the binary relation on tuples $t_1[\bar{A}] \approx t_2[\bar{B}]$, where $t_1 \in R$ and $t_2 \in S$. \square

Lemma 1. Let D be an instance and let m be the MD in Definition 15. An instance D' obtained by changing modifiable attribute values of D satisfies $(D, D') \models m$ iff for each equivalence class of T^\approx , there is a constant vector \bar{v} such that, for all tuples t in the equivalence class,

$$\begin{aligned} t'[\bar{C}] &= \bar{v} & \text{if } t \in R(D) \\ t'[\bar{E}] &= \bar{v} & \text{if } t \in S(D) \end{aligned}$$

where t' is the tuple in D' with the same identifier as t .

Proof: Suppose $(D, D') \models m$. By Definition 3, for each pair of tuples $t_1 \in R(D)$ and $t_2 \in S(D)$ such that $t_1[\bar{A}] \approx t_2[\bar{B}]$,

$$t'_1[\bar{C}] = t'_2[\bar{E}]$$

Therefore, if $T^\approx(\bar{t}_1, \bar{t}_2)$ is true, then t'_1 and t'_2 must be in the transitive closure of the binary relation expressed by $t'_1[\bar{C}] = t'_2[\bar{E}]$. But the transitive closure of this relation is the relation itself (because of the transitivity of equality). Therefore, $t'_1[\bar{C}] = t'_2[\bar{E}]$. The converse is trivial. \square

Proof of Theorem 5: We prove the specific case in which the MDs have the form

$$\begin{aligned} m_1 : R[\bar{A}] \approx_1 S[\bar{B}] \rightarrow R[C] \Leftarrow S[E] \\ m_2 : R[\bar{A}] \approx_1 S[\bar{B}] \wedge R[C] \approx_2 S[E] \rightarrow R[H] \Leftarrow S[I] \end{aligned}$$

A resolved instance is obtained after two updates. The pairs of tuples $t_1 \in R$ and $t_2 \in S$ such that the equality $t_1[C] = t_2[E]$ is imposed by the first update are those that satisfy $T^{\approx_1}(t_1, t_2)$, by lemma 1. The second update does not affect the values of the attributes $R[C]$ and $S[E]$, and the pairs of tuples $t_1 \in R$ and $t_2 \in S$ such that $t_1[H]$ and $t_2[I]$ are made equal are simply those that satisfy $T^{\approx_1}(t_1, t_2)$. Furthermore, the relation $T^{\approx_1}(t_1, t_2)$ subsumes the transitive closure of $t_1[\bar{A}] \approx_1 t_2[\bar{B}] \wedge t_1[C] \approx_2 t_2[E]$ before the first update, and so the net effect of the two updates on the values of $R[H]$ and $S[I]$ is to impose $t_1[H] = t_2[I]$ on pairs of tuples satisfying $T^{\approx_1}(t_1, t_2)$. Since $T^{\approx_1}(t_1, t_2)$ is computable is polynomial time, the result follows. \square