Matching Dependencies: Semantics and Query Answering

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) are used to declaratively specify the identification (or matching) of certain attribute values in pairs of database tuples when some similarity conditions on other values are satisfied. Their enforcement can be seen as a natural generalization of entity resolution. In what we call the *pure case* of MD enforcement, an arbitrary value from the underlying data domain can be used for the value in common that is used for a matching. However, the overall number of changes of attribute values is expected to be kept to a minimum. We investigate this case in terms of semantics and the properties of data cleaning through the enforcement of MDs. 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 query answering is investigated. Tractable and intractable cases depending on the MDs are identified and characterized.

Keywords: Databases, data cleaning, duplicate and entity resolution, integrity constraints, matching dependencies

1. INTRODUCTION

A relational database instance can be seen as a model of an external reality. As such, it may contain tuples and values in them that refer to the same external entity. In consequence, the database may be modeling the same entity in different forms, as different entities. This may be, most likely, not the intended representation. This is a problem that can be due to many different factors, among them, erroneous data entry, different sources of data, the use of different formats or semantics, etc. In this case, the database is considered to contain dirty data, and it must undergo a cleansing process.

For this particular kind of data dirtiness, namely unintended multiple representations of the same external entity, the cleaning process goes through two interlinked phases: detecting tuples (or values therein) that should be matched or identified; and next, doing the actual matching. This problem and process is usually called *entity resolution*, *data fusion*, *duplicate record detection*, etc. Cf. [16, 12] for some recent surveys, and [5] for more recent work in the area.

Quite recently, and generalizing entity resolution, [17, 18] introduced *matching dependencies* (MDs), which are declarative specifications of matchings of attribute values that should be made when certain conditions are satisfied. MDs help identify duplicate data and enforce their merging by exploiting semantic knowledge.

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 instance of a database predicate P:

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

The similarity of the names in the two tuples may be insufficient to establish that the tuples refer to the same person. This is because the last name is rather common, and only the first initial of one of the names is given. However, the similarities of their phone and address values indicate that the two tuples may be duplicates, and refer to the same individual. In this case, the names should be merged. This requirement is expressed by an MD which states that, if any two tuples from P have similar address and phone, then the names should be matched. This is expressed in MD notation by

$$P[Phone] \approx_{ph} P[Phone] \wedge P[Address] \approx_{a} P[Address] \rightarrow P[Name] \doteq P[Name],$$

where $\approx_{ph}, \approx_{a}$ are application (and attribute) dependent similarity relations for phone numbers and addresses. □

The identification in [17, 18] of this new class of dependencies and their declarative formulation have become important additions to data cleaning research. In this work, we further investigate MDs, starting by introducing our own refinement of the model-theoretic and dynamic semantics of MDs introduced in [18].

Any method of querying a dirty data source must address the issues of duplicate presence, detection, and resolution. The accuracy of query answering depends on them. Query

answering is typically done by first cleaning the data, by discarding or combining duplicate tuples and standardizing formats. The result of this process becomes a new, clean database where the entity conflicts have been resolved. The next step is directly querying the resulting database, and computing answers from it, as usual.

However, the entity resolution problem may have different *solution instances* (which we will simply call *solutions*). That is, different clean versions of the original database may arise and could be considered. In consequence, the query answers obtained via a particular solution may differ from those obtained using an alternative solution. We would like and expect that, given the initial, dirty instance, the acceptable query answers are those that are robust or invariant under the choice of individual clean instances. So, it becomes relevant to characterize those query answers that are invariant under the different (sensible) ways of cleaning the data.

If we assume that the relevant aspects of the entity resolution process to be applied to a given, possibly dirty instance, D, are captured by a set M of MDs associated to D's schema, then the issues we just mentioned can be properly taken care of by first providing the right *semantics for MDs*. As usual in many areas of data management and knowledge representation, this amounts to characterizing a class of *intended clean instances*, i.e. those that are acceptable results of enforcing M on D. This is a form of *model-theoretic semantics*, that in this paper we introduce and investigate. After doing this, the intended, *semantically clean* answers to a query are defined as those that persist across all the intended clean instances.

Characterizing and computing the clean query answers to queries is an interesting problem *per se*. However, it becomes crucial if we want to avoid computing and materializing all the possible clean instances, individually querying them, and collecting the answers in common. Ideally, one would like to obtain clean answers by querying nothing but the original dirty data source. Actually, this aspiration becomes a must when querying virtual data integration systems, where a central user does not have access or control over the sources, on which a material cleaning would have to be applied.

Similar problems have been investigated in the area of *consistent query answering* (CQA) [2], where, instead of MDs, classical integrity constraints (ICs) are considered. In that case, inconsistent database instances that violate the ICs are *repaired* in order to restore consistency. Also query rewriting methodologies for CQA have been proposed. With them CQA can be obtained by (appropriately) querying the initial inconsistent database. Cf. [6, 15, 7] for surveys of CQA. CQA in virtual data integration systems has also been investigated [8].

In this whole work, and in particular in the development of the semantics of MDs, we concentrate on what we call *the pure case* of MDs, the closest to the way MDs were introduced in [18]. In it, the values that can be chosen to match attribute values are arbitrarily taken from the underlying data domains. In Example 1, any name in common could be chosen for the matching, in principle. However, the total number of changes of attribute values that are due to the matchings should be minimized. As an alternative, entity resolution based on the use of *matching functions* that provide values for the matchings has also been investigated. In [5] entire tuples (records) are merged as opposed to individual attribute values. In [10], the combination of MDs and matching functions is formally developed.

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

1. We revisit the semantics of MDs introduced in [18], pointing out sensible and justified modifications that it requires.

2. A new semantics for MD satisfaction is then proposed and formally developed.

3. We formally define the intended solution instances for a given, initial instance, D_0, that is subject to a set of MDs. They are called *minimally resolved instances* (MRIs).

   MRIs are obtained through an iterative process that stepwise enforces the MDs until a stable instance is reached (i.e. no more MDs are applicable). The resulting instances minimally differ from D_0 wrt the *number of changes* of attribute values.

4. We introduce the notion of *resolved answers* to a query posed to D_0. They are the semantically clean answers, that are invariant under the MRIs.

5. We investigate the complexity of computing resolved answers. In particular, we establish a general upper bound of Π_2^P on data complexity. We also identify classes of MDs and queries for which *resolved query answering* is *NP*-hard. We also show some tractable cases.

6. We establish some comparisons between MRIs wrt MDs and database repairs wrt functional dependencies, showing in particular, that the former cannot be obtained from the latter.

This paper is organized as follows. Section 2 presents basic concepts and notations needed in the rest of the paper. Section 3 discusses the original MD semantics and proposes a revised version. In doing so, it introduces the notion of resolved instance for a given initial instance. Section 4 introduces the notion of resolved answer to a query, and investigates the computation and complexity of resolved answers. Section 5 presents some final conclusions, and points to ongoing and future work. This paper is a revised and extended version of [22]. Among other extensions, we include the proofs of the results stated there, and also the so far unpublished complexity upper bound for resolved query answering.

2. PRELIMINARIES

We consider a relational schema $\mathcal{S}$ that includes an enumerable infinite domain U. Database predicates in $\mathcal{S}$ have attributes, say A, whose domains, $Dom(A)$, are subsets of U. We sometimes refer to attribute A of a relation R by $R[A]$. We assume that (some) attributes have binary similarity relations $\approx_A$ on $Dom(A)$. They are reflexive and symmetric, and are treated as built-in predicates.

An instance D of $\mathcal{S}$ is a finite set of ground atoms (or tuples) 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. It can be seen as a unique value in an extra attribute that acts as a key for the relation

and is not subject to updates. In the following, it will be usually omitted, 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.

For a predicate $R \in \mathcal{S}$, $R(D)$ denotes instance D restricted to R (commonly referred to as the extension of R in D). If the ith attribute of predicate R is A, for a tuple $t = (c_1, \ldots, c_j) \in R(D)$, $t[A]$ denotes the value c_i. The symbol $t[\bar{A}]$ denotes the sequence whose entries are the values of the attributes in sequence of attributes $\bar{A}$.

A *matching dependency* [17], involving predicates $R(A_1, \ldots, A_n)$, $S(B_1, \ldots, B_p)$, is a symbolic expression (or a rule) of the form

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

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

The similarity relations, 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 intended to state an implicit universal quantifications over pairs of tuples t_1, t_2 for R and S, resp. The expression $\bigwedge R[A_i] \approx_{ij} S[B_j]$ on the LHS of the arrow 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.

There are two complimentary ways of interpreting this MD: statically and dynamically. According to the static interpretation, the MD is read as an implication, similar to a functional dependency (FD). It says that, if $\bigwedge R[A_i] \approx_{ij} S[B_j]$ holds, then, for each pair (A_i, B_j) such that $R[A_i] \doteq S[B_j]$ appears on the RHS, and for the same tuples t_1 and t_2, $t_1[A_i]$ and $t_2[B_j]$ are equal. The dynamic interpretation of the MD states that if the similarity conditions hold, such pairs of attribute values should be updated so that they become the same for t_1 and t_2. However, the attribute values to be used for this matching are left unspecified by (1). The static interpretation is useful for identifying dirty data, while the dynamic interpretation can be used as the basis for data cleaning procedure.

For abbreviation, we will sometimes write MDs as

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

where $\bar{A}$, $\bar{B}$, $\bar{C}$, and $\bar{E}$ represent the lists of attributes, $(A_1, ..., A_k)$, $(B_1, ..., B_k)$, $(C_1, ..., C_{k'})$, and $(E_1, ..., E_{k'})$, 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 $(\bar{C}, \bar{E})$.

In the dynamic interpretation, an MD involves update operations, to satisfy it. This requires a precise definition of satisfaction of an MD by a *pair* of database instances: An instance D and its updated instance D', as required by the MD.

Definition 1. [18] 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 $(D, D') \vDash_F m$, 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) $t'_R[\bar{A}] \approx t'_S[\bar{B}]$. □

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'), $(D, D') \vDash_F M$ means that $(D, D') \vDash_F m$, for every $m \in M$. Condition (b) in Definition 1 requires that the identification updates do not destroy the original similarities.

An instance D' is *stable* [18] for a set M of MDs if $(D', D') \vDash_F M$. Stability of an instance is a static concept, analogous to satisfaction by the instance D' of a set of FDs. 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 [18], for an instance D, if $(D, D') \vDash_F M$ for a stable instance D', then D' is expected to be reachable as a fix-point of an iteration of value identification updates that starts from D and is based on M.

3. THE MD SEMANTICS REVISITED

The requirement of keeping similarities after identification updates (cf. Definition 1) is a strong requirement that may lead to counterintuitive results.

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

$R(D)$	A	B	C
	a	c	g
	a	c	ksp

$S(D)$	E	F
	h	c
	msp	c

$$R[A] \approx R[A] \rightarrow R[C] \doteq R[C] \qquad (3)$$
$$R[C] \approx S[E] \rightarrow R[B] \doteq S[F] \qquad (4)$$

In this example, we assume that, for two strings s_1 and s_2, $s_1 \approx s_2$ holds when the edit distance d between s_1 and s_2 satisfies $d \leq 1$. To produce an instance D' satisfying $(D, D') \vDash_F M$, 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 values for attribute E in the tuples in S, due to 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'. □

Another problem with the semantics of MD satisfaction 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 $(D, D) \vDash_F M$, it is always possible, by choosing different common values, to produce a different instance D' such that $(D, D') \vDash_F M$. This is illustrated in the next example.

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

$R(D)$	A	B
	a	c
	a	c

Although D is stable, $(D, D') \models_F m$ is true for any D' where the values for attribute B in the two tuples are the same, e.g. d. □

3.1 MD satisfaction

We now propose a modified 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, M a set of MDs, and $\mathcal{P}$ be a set of pairs (t, G), where t is a tuple of D and G is an attribute of t.
(a) For a tuple $t_R \in R(D)$ and C an attribute of R, the value $t_R^D[C]$ is *modifiable wrt.* $\mathcal{P}$ if there exist $S \in \mathcal{R}$, $t_S \in S(D)$, an $m \in M$ of the form

$$R[\bar{A}] \approx S[\bar{B}] \;\rightarrow\; R[\bar{C}] \doteq S[\bar{E}],$$

and a corresponding pair (C, E) of $(\bar{C}, \bar{E})$ in m, such that $(t_S, E) \in \mathcal{P}$ and 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 wrt. $\mathcal{P} \setminus \{(t_S, E)\}$.

(b) The value $t_R^D[C]$ is *modifiable* if it is modifiable wrt. $\mathcal{V} \setminus \{(t_R, C)\}$, where $\mathcal{V}$ is the set of all pairs (t, G) with t a tuple of D and G an attribute of t. □

Definition 2 is recursive. The base case occurs when either case 1 applies (with any $\mathcal{P}$) or when there is no tuple/attribute pair in $\mathcal{P}$ that can satisfy part (a). Notice that recursion must terminate eventually, since the latter condition must be satisfied when $\mathcal{P}$ is empty, and each recursive call reduces the size of $\mathcal{P}$. The following example illustrates the definition.

Example 4. Consider $m: R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$ on schema $R[A, B]$, and the following instance. Assume that the only non-trivial similarities are $a_1 \approx a_2 \approx a_3$ and $b_1 \approx b_2$. Since $a_2 \approx a_3$ and $c_1 \neq c_3$, $t_2[B]$ and $t_3[B]$ are modifiable (base case). With case 2 of Definition 2, since $a_1 \approx a_2$, and $t_2[B]$ is modifiable, we obtain that $t_1[B]$ is modifiable.

$R(D)$	A	B
t_1	a_1	c_1
t_2	a_2	c_1
t_3	a_3	c_3
t_4	b_1	c_3
t_5	b_2	c_3

For $t_5[B]$ to be modifiable, it must be modifiable wrt. $\{(t_i, B) \mid 1 \leq i \leq 4\}$, and via t_4. According to case 2 of Definition 2, this requires $t_4[B]$ to be modifiable wrt. $\{(t_i, B) \mid 1 \leq i \leq 3\}$. However, this is not the case since there is no t_i, $1 \leq i \leq 3$, such that $t_4[A] \approx t_i[A]$. Therefore $t_5[B]$ is not modifiable. A symmetric argument shows that $t_4[B]$ is not modifiable. □

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:
(a) 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 $R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \doteq S[\bar{E}]$ 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 $t'_R[\bar{C}] = t'_S[\bar{E}]$.
(b) For any tuple $t_R \in R(D)$ and any attribute G of R, if $t_R[G]$ is not modifiable, then $t'_R[G] = t_R[G]$. □

Here, condition (b) captures a natural default condition of persistence of values: Only those that have to be changed are changed. Notice that now we are not requiring persistence of similarities. As before, we define *stable* instance D for M to mean $(D, D) \models M$. Except where otherwise noted, these are the notions of satisfaction and stability that will be used in the rest of this paper.

Example 5. (example 4 continued) The instances D' such that $(D, D') \models M$ are all those obtained from D by changing the values of $t_1[B]$, $t_2[B]$, and $t_3[B]$ to a common value, and leaving all other values unchanged. This is because these values are the only modifiable values, and they must be equal by condition (a) of Definition 3 and the given similarities. □

Condition (b) 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 $(D, D') \models M$.

Proof: 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 (a) 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 (a) 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 (b) 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 (a) 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 (a) 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. □

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 (a) 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 [18], 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 still be interested in the stable instances D' obtained through such an iterative enforcement of MDs. We are willing to relax the condition $(D, D') \models M$, but we will make sure, at each step k, 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 $D_1, D_2, ...D_n$ with: $(D, D_1) \models M$, $(D_1, D_2) \models M$,... $(D_{n-1}, D_n) \models M$, $(D_n, D') \models M$, and $(D', D') \models M$. □

Notice 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.

Resolved instances have been defined here by means of a *chase-like* iterative mechanism [1]. Our next result tells us that we can always obtain a resolved instance.

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

Proof: 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 $\{S_1, S_2, ...S_n\}$ 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. □

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

$$R[A] \approx R[A] \rightarrow R[B] \doteq R[B],$$
$$R[B] \approx R[B] \rightarrow R[C] \doteq R[C].$$

We assume that all pairs of distinct constants in R are dissimilar. The following instances, D_1 and D_2, are resolved instances of R:

$R(D_1)$	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 in the second tuple is not modifiable in D. This shows that some resolved instances cannot be obtained in a single update step, with updated instances as in Definition 4. □

Although Theorem 2 implies that, for any instance, the chase in Definition 4 terminates for some choice of update values, this is not necessarily the case for all choices of update values, as the next example shows.

Example 7. Consider the following set of MDs and instance D:

$$m_1\colon\ R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$$
$$m_2\colon\ R[B] \approx R[B] \rightarrow R[A] \doteq R[A]$$

$R(D)$	A	B
1	a	a
2	b	a
3	b	b

Here, the similarity relation is the equality. To specify the value that results when a pair of values are merged, we use the idempotent and commutative function f defined by $f(a,b) = c$, $f(c,b) = a$, and $f(c,a) = b$. Instance D is recovered after six updates starting from D:

$R(D_1)$	A	B
1	c	a
2	c	c
3	b	c

$\rightarrow$

$R(D_2)$	A	B
1	c	b
2	a	b
3	a	c

$\rightarrow$

$R(D_3)$	A	B
1	b	b
2	b	a
3	a	a

$\rightarrow$

$R(D_4)$	A	B
1	b	c
2	c	c
3	c	a

$\rightarrow$

$R(D_5)$	A	B
1	a	c
2	a	b
3	c	b

$\rightarrow$

$R(D_6)$	A	B
1	a	a
2	b	a
3	b	b

Therefore, this choice of update values results in an infinite chase sequence. Notice that the merge function in this example is non-associative, e.g. $f(a, f(b,c)) = a \neq c = f(f(a,b),c)$. For idempotent, commutative and associative merge functions, all the chase sequences finitely terminate [10, 11]. □

The notion of resolved instance is a first 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 *minimally resolved instance*. It uses as a measure of change the *number of values that were modified* to obtain the final clean instance. In Example 6, instance D_2 is a minimally resolved instance, whereas D_1 is not.

Definition 5. Let D be an instance.
(a) $T_D := \{(t, A) \mid t$ is the id of a tuple in D and A 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:
$S_{D,D'} := \{(t, A) \in T_D \mid f_D(t, A) \neq f_{D'}(t, A)\}$. □

Intuitively, $S_{D,D'}$ is the set of all "positions" within the instance such that the value at that position is changed when 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 with $Res(D, M)$ the set of minimal resolved instances of D wrt the set M of MDs. □

Example 8. Consider the instance below and the MD

R	A	B
	a_1	b_1

S	C	D
	c_1	d_1

$$R[A] \approx S[C] \rightarrow R[B] \doteq S[D].$$

Assuming that $a_1 \approx c_1$, this instance has two minimally 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

□

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. Since minimality is based on comparison of natural numbers (number of changes), from Theorem 2 we immediately obtain

Corollary 1. Every instance D has at least one MRI wrt a set of MDs. □

In CQA and database repairing, this kind of minimality has also been used [21, 20]. However, several other notions of minimality and comparison of instances have also been applied and investigated, most prominently the one based on tuple insertion/deletion and minimal symmetric set difference [2]. Cf. [7, section 2.5] for a survey of repair semantics.

In this subsection, we defined the MRIs, which we use as our model of a clean database instance. They capture the dynamic interpretation of MDs. This observation leads us to wonder if we could have taken a more traditional, static approach, in which the MDs are interpreted as integrity constraints and the clean instances as database repairs. We now show by means of an example that such an approach is not appropriate in the MD context.

Example 9. Consider the following instance of a relation R and MD:

R	A	B
	a	b
	a	d

$$R[A] = R[A] \rightarrow R[B] \doteq R[B].$$

Here, the similarity relation is the equality, and we assume, as usual, that for pairs of distinct constants in R the inequality holds.

Suppose we view the MD as a functional dependency to be satisfied by R, replacing $\doteq$ by $=$. The given instance would be inconsistent. Furthermore, consider the *repairs* (in the sense of CQA) that would be obtained via attribute modification [21, 28, 9, 20, 6], and minimality as in Definition 6.

It follows immediately from Definition 6 that all MRIs are repairs. However, not every repair would be a MRI, because one way of repairing the instance would be to change one of the values in the A column, e.g. to b. In the context of duplicate resolution this would be undesirable. Actually, the appropriate way to repair R would be to set the values in the B column to a common value. □

In failing to restrict the allowed updates, an approach that defines MDs as traditional integrity constraints will lead to undesirable repairs, and will not provide an appropriate semantics for MD-based entity resolution.

The requirement of Definition 6 of minimizing the number of changes leading to an MRI can be relaxed, to allow MRIs whose change is within some percentage of the minimum without affecting any of the results presented here. This might be a more appropriate definition in certain duplicate resolution settings.

In this work we are investigating what we could call "the pure case" of MD-based entity resolution. It adheres to the original semantics outlined in [18], which does not specify how the matchings are to be done, but only which values must be made equal. That is, the MDs have implicit existential quantifiers (for the values in common). The semantics we just introduced formally captures this pure case. We find situations like this in other areas of data management, e.g. with referential integrity constraints, tuple-generating dependencies in general [1], schema mappings in data exchange [4], etc. A non-pure case that uses matching functions to realize the matchings as prescribed by MDs is investigated in [10, 11, 3].

4. RESOLVED QUERY ANSWERS

Having defined the MRIs as the intended instances for an MD-based entity resolution, we are in position to define the *resolved answers to a query*. Intuitively, these are the answers that are true in all MRIs. This is analogous to CQA, where a consistent answer to a query is defined as being true in all minimal repairs of a database that violates a set of integrity constraints [2].

Let $\mathcal{Q}(\bar{x})$ be a query expressed in the first-order language $L(\mathcal{S})$ associated to schema $\mathcal{S}$. Here, $\bar{x}$ represents the sequence of free variables, say $\bar{x} = x_1 \cdots x_n$, whose combinations of values from the database instance form the answers to the query. More precisely, $\bar{c} = c_1 \cdots c_n$, with $c_i \in U$, is an answer to $\mathcal{Q}$ from instance D, denoted $D \models \mathcal{Q}[\bar{c}]$, iff (the FO formula representing) $\mathcal{Q}(\bar{x})$ becomes true in D when the variables x_i take the values c_i.

A query is *conjunctive* when it is of the form

$$\mathcal{Q}(\bar{x}) \colon \exists \bar{y}(A_1(\bar{x}_1) \wedge \cdots \wedge A_k(\bar{x}_k)),$$

where the $A_i(\bar{x}_i)$ are atoms of $L(\mathcal{S})$, and each of the variables in $\bar{x}$ appears in some of the $\bar{x}_i$, but among those not in $\bar{y}$.

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

Example 10. (example 8 continued) The set of resolved answers to the query $\mathcal{Q}_1(x, y) \colon R(x, y)$ is empty since there are is no tuple in the intersection of all the extensions of R in minimal resolved instances. On the other hand, the set of resolved answers to $\mathcal{Q}_2(x) \colon \exists y R(x, y)$ is $\{\langle a_1 \rangle\}$. □

The notion of resolved answer is based on the class of minimally resolved instances. It is not difficult to show that this class can be quite large, actually exponentially large in the size of the original instance [23]. As a consequence, computing, materializing and querying all the MRIs to do resolved query answering should be replaced by more efficient alternatives whenever possible. This requires a a better understanding of the intrinsic complexity of resolved query answering. Actually, in Section 4.1 we will investigate the complexity of the problem of computing the resolved answers, which we now formally introduce, as a decision problem [24].

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

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

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. □

Notice that here, the set of MDs and the query are fixed, and the instances for the decision problem are the pairs formed by a database instance and a potential resolved answer from D. As a consequence, the complexity of the RAP problem is *data complexity*, i.e. measured in terms of the size of instance D [1].

4.1 Computing resolved answers

In this section, we consider the complexity of the problem $RA_{\mathcal{Q},M}$ just introduced. In this direction, it is useful to associate a graph with a set of MDs. For this we need to introduce a few notions first.

Definition 9. A set M of MDs is in *standard form* if no two MDs in M have the same expression on the left-hand side of the arrow. □

Notice that any set M of MDs can be put in standard form by replacing a subset of M of the form

$$\{R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}_1] \doteq S[\bar{E}_1], \quad \ldots, \quad R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}_n] \doteq S[\bar{E}_n]\}$$

by the single MD

$$R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \doteq S[\bar{E}],$$

where the set of corresponding pairs of attributes of $(\bar{C}, \bar{E})$ is the union of those of $(\bar{C}_1, \bar{E}_1)$, ...$(\bar{C}_n, \bar{E}_n)$. From now on, we will assume that all sets of MDs are in standard form.

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

Definition 10. Let M be a set of MDs in standard form. (a) The *MD-graph* of M, denoted $MDG(M)$, is a directed graph with a vertex labeled with m for each $m \in M$, and with an edge from m_1 to m_2 iff $RHS(m_1) \bigcap LHS(m_2) \neq \emptyset$. (b) A set of MDs whose MD-graph contains edges is called *interacting*. Otherwise, it is *non-interacting*. □

Example 11. Consider the set M of MDs in standard form:

$$\begin{aligned} m_1 &: \; R[A] \approx S[B] \rightarrow R[C] \doteq S[D], \\ m_2 &: \; R[C] \approx S[D] \rightarrow R[A] \doteq S[B] \wedge R[A] \doteq S[F] \wedge R[C] \doteq S[E], \\ m_3 &: \; S[E] \approx S[B] \rightarrow S[F] \doteq S[F]. \end{aligned}$$

Its MD-graph is shown in Figure 1. It shows that M is an interacting set of MDs. □

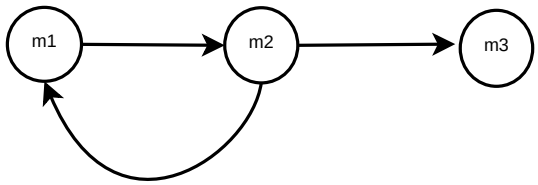

Figure 1: MD-Graph

As a first approach to comparing the tractability of resolved query answering for different sets M of MDs, we consider the tractability of retrieving the resolved answers, i.e. of the decision problem $RA_{\mathcal{Q},M}$, for a particular class of conjunctive queries, namely *single atom queries*. They are of the form $\mathcal{Q}(\bar{x})\colon\ \exists\bar{y}R(\bar{z})$, where R is a database predicate of schema $\mathcal{S}$, and $\bar{x} = \bar{z} \smallsetminus \bar{y}$. If we have intractability for that class of queries, we will have the same for the broader class of conjunctive queries.

It is not difficult to verify that, for non-interacting sets of MDs, the resolved answers to such queries are retrievable in polynomial time. This follows from the fact that a resolved instance is obtained by applying the MDs once to the instance. The only values that will be left unchanged after this update in all MRIs are those that are not part of a set of duplicates, and, by the minimality condition on MRIs, those that occur more frequently than any other value in the set of duplicates to which they belong. The resolved answers correspond to those tuples such that none of the values returned by the query can change in an MRI.

We now turn to the simplest case of interacting MDs, namely a set M of two MDs such that $MDG(M)$ has a single directed edge from one vertex to the other.

Definition 11. An ordered pair (m_1, m_2) of MDs is a *linear pair* of MDs if $MDG(\{m_1, m_2\})$, the MD graph of $\{m_1, m_2\}$, has a single edge, and from m_1 to m_2. □

Now we will investigate the complexity of $RA_{\mathcal{Q},M}$ when M is a linear pair of MDs. Actually, as we will show below (cf. Theorem 3), this problem may become intractable already for this simple case. In order to obtain this result we make the assumption that, for all similarity operators, there exists an infinite set of pairwise dissimilar values. This is not a strong assumption since attribute domains are usually infinite. However, this assumption can be weakened by requiring that a *sufficiently large* number of dissimilar elements exist.

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 in common are also similar in D'. We call *intended similarities* those similarities of D' which hold in D or which are implied by $(D, D') \models M$. 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 values, and may not occur in all D' satisfying $(D, D') \models M$. This latter kind may unnecessarily enable MDs, complicating the entity resolution process. As a consequence, distinguishing them from the intended ones becomes important.

Example 12. Consider the two-attribute predicate $R(A, B)$ belonging to a schema also containing $S(C, D)$, and the linear pair (m_1, m_2) formed by

$$m_1:\quad R[A] = R[A] \quad \rightarrow \quad R[B] \doteq R[B],$$
$$m_2:\quad R[B] = R[B] \quad \rightarrow \quad S[C] \doteq S[C].$$

Consider the following initial extension for R

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

The application of m_1 might lead to the following updated instance

R	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 $t_2[B] = t_4[B]$. These accidental similarities are also "accidentally" enabling MD m_2 for unintended tuples. □

In the case of interacting MDs, accidental similarities are a source of intractability for the computation of resolved answers. 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 3. The RAP problem can be intractable for conjunctive queries and linear pairs of MDs. More precisely, for the the query $\mathcal{Q}(x, z)\colon\ \exists y R(x, y, z)$, and the following set M of MDs

$$R[A] \approx R[A] \quad \rightarrow \quad R[B] \doteq R[B]$$
$$R[B] \approx R[B] \quad \rightarrow \quad R[C] \doteq R[C]$$

$RA_{\mathcal{Q},M}$ is *NP*-hard. □

In this result, *NP*-hardness is defined in terms of Turing (or Cook) reductions as opposed to many-one (or Karp) reductions [24, 26]. This former notion of hardness is weaker or more general than the latter, in the sense that *NP*-hardness under many-one reductions implies *NP*-hardness under Turing reductions, but the converse, although widely conjectured not to hold, is an open problem. However, for Turing reductions, it is still true that there is no efficient algorithm for an *NP*-hard problem, unless $P = NP$.

Our use of Turing reductions also explains the *NP*-hardness result, as opposed to *co-NP*-hardness (based on many-one reductions) that one usually establishes in cases like this: A certificate showing that a tuple *is not* a resolved answer would be an MRI that does not return the tuple as a usual answer to the query. Under Turing reductions, the notions of *NP*-hardness and *co-NP*-hardness coincide.

The proof given below employs an oracle machine that uses an oracle for RAP to decide the satisfiability of a MONOTONE 3-SAT formula F. This is used instead of a Karp reduction, because there are several different tuples that must

be present in a resolved instance for F to be satisfiable. This requires several instances of RAP to be decided to determine satisfiability of F. Each of those tuples is associated with a clause of F, and the presence of all of them signifies that there is a way to satisfy all clauses without assigning both true and false to any variable.

Proof of Theorem 3: The proof is by Turing reduction from MONOTONE 3-SAT. Given an instance F of MONOTONE 3-SAT with clauses c_1, c_2,...c_n, let D be an instance of R with tuples t_1, t_2,...t_{3n}. The sets $\{t_1, t_2, t_3\}$, $\{t_4, t_5, t_6\}$,... are called 3-blocks. For $0 \leq i < n$, $t_{3i+1}[A] = t_{3i+2}[A] = t_{3i+3}[A] = k_i$, where 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 $t_{3i+1}[C] = t_{3i+2}[C] = t_{3i+3}[C] = a$. In this case, the set $\{t_{3i+1}, t_{3i+2}, t_{3i+3}\}$ is called a positive 3-block. If c_i is a negative clause, then $t_{3i+1}[C] = t_{3i+2}[C] = t_{3i+3}[C] = b$. In this case, the set $\{t_{3i+1}, t_{3i+2}, t_{3i+3}\}$ is called a negative 3-block. The values in the B column consist of a set S of pairwise dissimilar values, one for each variable in F. The values of $t_{3i+1}[B]$, $t_{3i+2}[B]$, and $t_{3i+3}[B]$ 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. Clearly, this implies that all MRIs have this form when F is satisfiable.

In a satisfying assignment to F, there is a literal for each clause in F that is made true by the assignment. In the first update, 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 positive 3-blocks are dissimilar to those chosen for the negative 3-blocks.

In the first update, if a value v is chosen as the common B attribute value for a positive 3-block, update the C attribute value for all tuples with v as their B attribute value to a if it is not already a. Similarly, if a value v is chosen as the common B attribute value for a negative 3-block, update the C attribute value for all tuples with v as their B attribute value to b if it is not already b. All other modifiable values in the C column can be updated to some arbitrary value.

After the second (and final) update, the C attribute values of the tuples in each 3-block must be the same. By choice of first update value for the C column, if there is a tuple in a positive (negative) 3-block that does not have a (b) as its C attribute value, then its C attribute value is modifiable after the first update. By choice of first update value for the B column, the update value for any modifiable C attribute value in any tuple in a positive 3-block can be chosen independently of that for any modifiable C attribute value in any tuple in a negative 3-block. 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). Suppose update values are chosen so as to achieve minimal change in the B column in the first update. Then it must be the case that, after the second update, values in the C column that were originally distinct will have been set to a common value. Specifically, pairs of 3-blocks whose C attribute values were originally distinct will have had their C attribute values set to a common value.

To determine whether the instance F is satisfiable, the oracle Turing machine first constructs instance D. It then poses the query $(D, (k_i, a))$ to the oracle if c_i is positive and the query $(D, (k_i, b))$ if c_i is negative, for $0 \leq i < n$. If the oracle answers yes in all cases, then the machine accepts. Otherwise, it rejects. □

It can be seen from the proof of Theorem 3, that the intractability of RAP for the set of MDs in it has its origin in the possibility of accidental similarities between values for attribute B. It is the intractability of determining whether or not accidental similarities can occur between certain values (which is actually established in the proof of the theorem) that makes the RAP intractable in this case.

Actually, for a set M of MDs like those in Theorem 3, each of the resolved instances for a given instance D can be obtained in at most two chase steps (this follows from Lemma 1 below, and the fact that M has an augmented MD graph of depth 1). However, we can still have exponentially many resolved instances: the total number of different choices of update values for all the sets of duplicate values taken together may be exponential.

The sources of complexity are then, the possibly exponential number of (minimally) resolved instances, and the minimality test. Actually, the latter is the main source: If RAP were defined on the basis of the resolved -not necessarily minimal- instances, it would be tractable for the MDs considered. This is because all we need to know are the sets of duplicate values and the possible update values for each set. Actually, the minimality as a source of complexity is also reflected in the fact that deciding the minimality of a resolved instance is *NP*-hard. This follows from the proof of Theorem 3.

In Theorem 3, the chosen query is also relevant for the intractability, because it returns values for an attribute that is updated more than once ($R[C]$ in that case). Updates on those values are affected by the accidental similarities generated during the previous update.

For some cases of linear pairs, resolved query answering is tractable 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.

Example 13. For the set of MDs

$$R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$$
$$R[A] \approx R[A] \wedge R[B] \approx R[B] \rightarrow R[C] \doteq R[C],$$

the resolved answers to any single-atom query involving predicate R can be retrieved in polynomial time, making RAP tractable for those queries.

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 4. For any pair (m_1, m_2) of linear MDs of the form

$$m_1 : \ R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \doteq S[\bar{E}]$$
$$m_2 : \ R[\bar{A}] \approx S[\bar{B}] \wedge R[\bar{C}] \approx S[\bar{E}] \rightarrow R[\bar{H}] \doteq S[\bar{I}]$$

the resolved answers to any single-atom query are retrievable in polynomial time.

Proof: The given set of MDs has the same MRIs as the MD

$$m' : \ R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \doteq S[\bar{E}] \wedge R[\bar{H}] \doteq S[\bar{I}]$$

which obviously satisfies the theorem. To see this, notice that no tuple $t_1 \in R$ (S) can have its values for the attributes in $RHS(m_1)$ or $RHS(m_2)$ updated by application of m_1 and m_2 if there is no tuple $t_2 \in S$ (R) such that $t_1[\bar{A}] \approx t_2[\bar{B}]$ ($t_2[\bar{A}] \approx t_1[\bar{B}]$). On the other hand, in all updates subsequent to the first, any pair of tuples satisfying $R[\bar{A}] \approx S[\bar{B}]$ must have their values matched according to $R[\bar{C}] \doteq S[\bar{E}]$ and $R[\bar{H}] \doteq S[\bar{I}]$. □

This result can be extended to a broader class of conjunctive queries by means of a query rewriting methodology [23].

4.2 A general upper bound

In this section, we show that, for certain classes of MDs and arbitrary queries that are evaluable in polynomial time, the time complexity of RAP is bounded above by the second level of the polynomial hierarchy [27].

The sets of MDs we consider satisfy a restriction that guarantees that the update process will terminate after a number of steps that does not depend on the size of the data. To express this restriction, we require an extension of MD-graphs which we call *augmented MD-graphs*.

Definition 12. Let M be a set of MDs on schema $\mathcal{S}$.
(a) The symmetric binary relation $\doteq_r$ relates attributes $R[A]$, $S[B]$ of $\mathcal{S}$ whenever there is an MD m in M where $R[A] \doteq S[B]$ appears to the right of the arrow of m.
(b) The *attribute closure* of M is the reflexive and transitive closure of $\doteq_r$.
(c) $E_{R[A]}$ denotes the equivalence class of attribute $R[A]$ in the attribute closure of M. □

Definition 13. Let M be a set of MDs in standard form. The *augmented MD-graph* of M, denoted $AMDG(M)$, is a directed graph with a vertex labeled with m for each $m \in M$, and with an edge from m to m' iff there is an attribute, say $R[A]$, with $R[A] \in RHS(m)$ and $E_{R[A]} \cap LHS(m') \neq \emptyset$. □

Example 14. (example 11 continued) The set of MDs has the augmented MD-graph shown in Figure 2. It must have all the edges in the original MD-graph has. In addition, it has an edge from m_1 to m_3, because $S[C] \in E_{R[C]}$; and also an edge from m_3 to m_1, because $R[A] \in E_{S[F]}$. □

Lemma 1. Let M be a set of MDs on schema $\mathcal{S}$ such that $AMDG(M)$ is a directed acyclic graph (DAG). Let D be an instance for schema $\mathcal{S}$. Then, any sequence of updates to D according to M (and consistent with Definition 4) terminates with a resolved instance after at most $d+1$ steps, where d is the maximum length of a path in $AMDG(M)$.

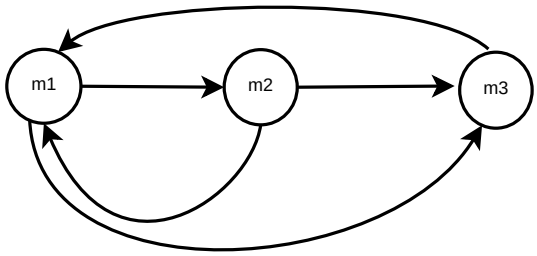

Figure 2: Augmented MD-Graph

Proof: Define the *depth* of an equivalence class E of the attribute closure of M as the maximum length of a path ending in an MD m with an element of E in $RHS(m)$.

Let A be an attribute such that the depth of E_A is d. We show by induction on the depth that no value of A for instance D can change after $d+1$ updates.

Let M_{E_A} denote the set of MDs m such that there is an attribute of E_A in $RHS(m)$. In the $d = 0$ case, all $m \in M_{E_A}$ satisfy the property that the values of attributes in $LHS(m)$ are always unmodifiable. The only way that a value of A can be modifiable is if there is a pair of tuples t_1 and t_2 that satisfy the condition of an MD m and a pair of (not necessarily distinct) attributes $B, C \in E_A$ such that $B \doteq C$ is a corresponding pair of m and $t_1[B]$ is unequal to $t_2[C]$. However, this cannot happen after the first update, since the values of all attributes in $LHS(m)$ are unmodifiable.

Suppose the statement is true for $d < k$. Let A be an attribute such that the depth of E_A is k. Then, all $m \in M_{E_A}$ satisfy the property that the values of all attributes of $LHS(m)$ are unmodifiable after the k^{th} update. A similar argument to that used for the $d = 0$ case shows that the values of A are not modifiable after the $(k+1)^{th}$ update. The lemma follows. □

We now state the main result of this section. For it, we will make the assumption that, during the update process, any string introduced into an instance from outside the active domain is bounded in length by a fixed polynomial in the maximum length of the values in the active domain. It is reasonable to make this *bounded length assumption* in practice, because duplicates would never be replaced by a value that differed widely from them in length.

Theorem 5. Let M be a set of MDs such that $AMDG(M)$ is a DAG. For any polynomial time evaluable query $\mathcal{Q}$, the decision problem $RA_{\mathcal{Q},M}$ is in Π_2^P.

Proof: The problem is decidable in polynomial time by an alternating Turing machine T that undergoes a universal phase followed by an existential phase. Let $(D, \bar{a})$ be the input to T. In the universal phase, T produces a resolved instance D' by applying the MDs to D, guessing the update values at each step. It then poses $\mathcal{Q}$ to D. If $\bar{a}$ is in the answer, T accepts. Otherwise, T enters an existential phase, and produces a resolved instance D'' as in the universal phase. If D'' has fewer changes than D', T accepts. Otherwise T rejects. The polynomial runtime is guaranteed by Lemma 1 and the bounded length assumption. □

Notice that this result can be applied to FO queries and Datalog queries, including those with stratified negation [1].

5. CONCLUSIONS

In this paper we have proposed a revised semantics for matching dependency (MD) satisfaction wrt the one originally proposed in [18]. On this basis we defined 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. In particular, we established the existence of MRIs.

We undertook the first steps in the investigation of the complexity of computing the RAs, and derived some first results in this direction. We showed that the problem may be intractable for simple cases of queries and MDs. However, we identified some non-trivial cases where tractability is guaranteed.

In our ongoing work [23], we are deriving syntactic criteria on MDs for identifying what we called *easy* and *hard sets of MDs*. We are also developing, in the easy cases, query rewriting methods for obtaining the RAs to much more general conjunctive queries than those considered in this paper.

We are also investigating the possibility of specifying the minimally resolved instances, and doing resolved query answering, by means of *answer set programs* (ASPs) [25, 13] with weak constraints [14]. The latter are used to impose the minimality condition on value changes on the stable models of the ASP that represent the minimally resolved instances.

ASPs have already been proposed for ER under MDs *and matching functions* (MFs) that are applied when merging two values [3]. The MFs are assumed to be idempotent, commutativity and associative, which always guarantees chase termination [10, 11], and then, finite stable models for the ASPs. In our case, we do not have that guarantee (cf. Example 7). As a consequence, ASP specifications should be possible when all chase sequences terminate, e.g. under the conditions of Lemma 1. Ongoing work addresses the identification of other cases of syntactic conditions on MDs that guarantee termination for every chase sequence.

The semantics of MDs in this paper is dynamic in the sense that it involves an update operation. Edit rules for master data is another area of data cleaning research in which a dynamic semantics has been proposed [19]. In contrast to duplicate resolution, which prescribes matchings of values without specifying the update value, edit rules require that a value be updated to a specific value contained in master data. Another difference is that with edit rules, updates to a tuple t are done on the basis of equality of values in t with values in master data, and these values of t are assumed to be certain and cannot change. With MDs, the values on the basis of which a matching was made are allowed to change as a result of subsequent updates. Indeed, this can happen whenever we have interacting MDs.

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 [10, 11, 3]. 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, 1995.

[2] M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *Proc. PODS*, ACM Press, 1999, pp. 68-79.

[3] Z. Bahmani, L. Bertossi, S. Kolahi and L. Lakshmanan. Declarative Entity Resolution Via Matching Dependencies and Answer Set Programs. Proc. KR 2012.

[4] P. Barcelo. Logical Foundations of Relational Data Exchange. *SIGMOD Record*, 2009, 38(1):49-58.

[5] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. Euijong Whang and J. Widom. Swoosh: A Generic Approach to Entity Resolution. *VLDB Journal*, 2009, 18(1):255-276.

[6] L. Bertossi. Consistent Query Answering In Databases. *ACM Sigmod Record*, 2006, 35(2):68-76.

[7] L. Bertossi. *Database Repairing and Consistent Query Answering*, Morgan & Claypool, Synthesis Lectures on Data Management, 2011.

[8] L. Bertossi and L. Bravo. Consistent Query Answers In Virtual Data Integration Systems. In *Inconsistency Tolerance*, Springer LNCS 3300, 2004, pp. 42-83.

[9] L. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. The Complexity And Approximation of Fixing Numerical Attributes in Databases under Integrity Constraints. *Information Systems*, 2008, 33(4):407-434.

[10] L. Bertossi, S. Kolahi, and L. Lakshmanan. Data Cleaning and Query Answering with Matching Dependencies and Matching Functions. In *Proc. ICDT*, ACM Press, 2011.

[11] L. Bertossi, S. Kolahi and L. Lakshmanan. Data Cleaning and Query Answering with Matching Dependencies and Matching Functions. To appear in *Theory of Computing Systems* journal, Springer, 2012.

[12] J. Bleiholder and F. Naumann. Data Fusion. *ACM Computing Surveys*, 2008, 41(1):1-41.

[13] G. Brewka, T. Eiter and M. Truszczynski. Answer Set Programming at a Glance. *Comm. of the ACM*, 2011, 54(12), pp. 93-103.

[14] F. Buccafurri, N. Leone and P. Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 2000, 12(5):845-860.

[15] J. Chomicki. Consistent Query Answering: Five Easy Pieces. In *Proc. ICDT*, Springer LNCS 4353, 2007, pp. 1-17.

[16] A. Elmagarmid, P. Ipeirotis and V. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 2007, 19(1):1-16.

[17] W. Fan. Dependencies Revisited for Improving Data Quality. In *Proc. PODS*, ACM Prees, 2008, pp. 159-170.

[18] W. Fan, X. Jia, J. Li and S. Ma. Reasoning about Record Matching Rules. In *Proc. VLDB*, 2009, pp. 407-418.

[19] W. Fan, J. Li, S. Ma, N. Tang, W. Yu. Towards Certain Fixes with Editing Rules and Master Data. In *Proc. VLDB Endowment*, 2010, pp. 173-184.

[20] S. Flesca, F. Furfaro and F. Parisi. Querying and

Repairing Inconsistent Numerical Databases. *ACM Trans. Database Syst.*, 2010, 35(2).

[21] 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*, Springer LNCS 2250, 2001, pp. 561-578.

[22] J. Gardezi, L. Bertossi and I. Kiringa. Matching Dependencies with Arbitrary Attribute Values: Semantics, Query Answering and Integrity Constraints. In *Proc. of the International Workshop on Logic in Databases (LID 2011)*, ACM Press, 2011.

[23] J. Gardezi and L. Bertossi, L. Query Answering under Matching Dependencies for Data Cleaning: Complexity and Algorithms. CorrArXiv paper cs.DB/1112.5908. 2012.

[24] M. Garey and D. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.

[25] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9(3/4), pp.365-386.

[26] O. Goldreich. *Computational Complexity*. Cambridge Univ. Press, 2008.

[27] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[28] J. Wijsen. Database Repairing Using Updates. *ACM Transactions on Database Systems*, 2005, 30(3):722-768.