Data Cleaning and Query Answering with Matching Dependencies and Matching Functions

Leopoldo Bertossi[1], **Solmaz Kolahi**[2], and **Laks V. S. Lakshmanan**[2]

[1] Carleton University, Ottawa, Canada. bertossi@scs.carleton.ca
[2] University of British Columbia, Vancouver, Canada. {solmaz,laks}@cs.ubc.ca

Abstract. Matching dependencies were recently introduced as declarative rules for data cleaning and entity resolution. Enforcing a matching dependency on a database instance identifies the values of some attributes for two tuples, provided that the values of some other attributes are sufficiently similar. Assuming the existence of matching functions for making two attribute values equal, we formally introduce the process of cleaning an instance using matching dependencies, as a chase-like procedure. We show that matching functions naturally introduce a lattice structure on attribute domains, and a partial order of semantic domination between instances. Using the latter, we define the semantics of clean query answering in terms of certain/possible answers as the greatest lower bound/least upper bound of all possible answers obtained from the clean instances. We show that clean query answering is intractable in general. Then we study queries that behave monotonically w.r.t. semantic domination order, and show that we can provide an under/over approximation for clean answers to monotone queries. Moreover, non-monotone positive queries can be relaxed into monotone queries.

1 Introduction

Matching dependencies (MDs) in relational databases were recently introduced in [19] as a means of codifying a domain expert's knowledge that is used in improving data quality. They specify that a pair of attribute values in two database tuples are to be matched, i.e., made equal, if similarities hold between other pairs of values in the same tuples. This is a generalization of entity resolution [17], where basically full tuples have to be merged or identified on the basis that they seem to refer to the same entity of the outside reality. This form of data fusion [12] is important in data quality assessment and in data cleaning.

Matching dependencies were formally studied in [20], as semantic constraints for data cleaning and were given a model-theoretic semantics. The main emphasis in that paper was on the problem of entailment of MDs and on the existence of a formal axiom system for that task.

MDs as presented in [20] do not specify *how* the matching of attribute values is to be done. In data cleaning, the user, on the basis of his or her experience and knowledge of the application domain, may have a particular methodology or heuristic for enforcing the identifications. *In this paper we investigate MDs* *in the context of matching functions*. These are functions that abstract the implementation of value identification. Rather than investigate specific matching functions, we explore a class of matching functions satisfying certain natural and intuitive axioms. With these axioms, matching functions impose a lattice-theoretic structure on attribute domains. Intuitively, given two input attribute values that need to be made equal, the matching function produces a value that *contains* the information present in the two inputs and *semantically dominates* them. We show that this semantic domination partial order can be naturally lifted to tuples of values as well as to database instances as sets of tuples. The following example illustrates the role of matching functions.

Example 1. Consider the database instance D_0 of schema $R(name, phone, address)$, shown below. Assume there is a matching dependency stating that if for two tuples the values of name and phone are similar, then the value of address should be made identical. This MD can be formally written as:

$$R[name, phone] \approx R[name, phone] \rightarrow R[address] \rightleftharpoons R[address].$$

Consider a similarity relation that indicates the values of name and phone are similar for the two tuples in this instance. To enforce the matching dependency, we create another instance D_1 in which the value of address for two tuples is the result of applying a matching function on the two previous addresses. This function combines the information in those address values, and thus D_1 semantically dominates D_0.

D_0	*name*	*phone*	*address*
	John Doe	(613)123 4567	Main St., Ottawa
	J. Doe	123 4567	25 Main St.

$$\Downarrow$$

D_1	*name*	*phone*	*address*
	John Doe	(613)123 4567	25 Main St., Ottawa
	J. Doe	123 4567	25 Main St., Ottawa

We can continue this process in a chase-like manner if there are still other MD violations in D_1. ■

The framework of [20] leaves the implementation details of data cleaning process with MDs completely unspecified and implicitly leaves it to the application on hand. We point out some limitations of the proposal in [20] for purposes of cleaning dirty instances in the presence of multiple MDs. We also show that a formulation of the formal semantics of satisfaction and enforcement of MDs that incorporates matching functions remedies this problem. In giving such a formulation, we revisit the original semantics for MDs proposed in [20], propose some changes and investigate their consequences. More precisely, we define *intended clean instances*, those that are obtained through the application of the MDs in a chase-like procedure. We further investigate properties of this procedure in relation to the properties of the matching functions, and show that, in general, the chase procedure produces several different clean instances, each of which semantically dominates the original dirty instance.

We then address the problem of query answering over a dirty instance, where the MDs do *not* hold. We take advantage of the semantic domination order between instances, and define *clean answers* by specifying a tight lower bound (corresponding to certain answers) and a tight upper bound (corresponding to possible answers) for all answers that can be obtained from any of the possibly many clean instances. We show that computing the exact bounds is intractable in general. However, in polynomial time we can generate an under-approximation for certain answers as well as an over-approximation for possible answers for queries that behave *monotonically* w.r.t. the semantic domination order.

We argue that monotone queries provide more informative answers on instances that have been cleaned with MDs and matching functions. We therefore introduce new relational algebra operators that make use of the underlying lattice structure on the domain of attribute values. These operators can be used to *relax* a regular positive relational algebra query and make it monotone w.r.t. the semantic domination order.

Recently, Swoosh [9] has been proposed as generic framework for entity resolution. In entity resolution, whole tuples are identified, or merged into a new tuple, whenever similarities hold between the tuples on some attributes. Accordingly, the similarity and matching functions work at the tuple level. Given their similarity of purpose, it is interesting to seek the relationship between the frameworks of MDs and of Swoosh. We address this question in this paper.

In summary, we make the following contributions:

1. We identify some limitations of the original proposal of MDs [20] w.r.t. the application of data cleaning in the presence of multiple MDs, and show how they can be overcome by considering MDs along with matching functions. We provide a precise formal and operational semantics for MD enforcement with matching functions. It appeals to an appropriate notion of *chase* with MDs.
2. We study matching functions in terms of their properties, captured in the form of certain intuitive and natural axioms. Matching functions induce a lattice framework on attribute domains which can be lifted to a partial order over instances, that we call *semantic domination*. The semantics of MD enforcement is compatible with, and relies on, the semantic domination structure.
3. We formally characterize clean query answering over a dirty instance w.r.t. a set of MDs. We define appropriate notions of certain and possible answer. We establish that computing clean answers is intractable in general.
4. We define a notion of monotone query that is based on semantic domination. We also introduce and investigate new monotone relational operators that are defined on the lattice structure of the data domain. In particular, we use them to provide a notion of query relaxation.
5. For queries that are monotone w.r.t. the semantic domination relation (which happen to be still intractable), we develop a polynomial time heuristic procedure for obtaining under- and over-approximations of clean query answers.

6. We demonstrate the power of the framework of MDs and of our chase procedure for MD application by reconstructing the general form of Swoosh, and also its special and common case called *the union and merge class*. This is all done by introducing appropriate matching dependencies with matching functions.

The paper is organized as follows. In Section 2, we provide necessary background on matching dependencies as originally introduced. We introduce matching functions and the notion of semantic domination in Section 3. Then we define the data cleaning process with MDs in Section 4. We explore the semantic of query answering in Section 5. In Section 6, we introduce and study a notion of monotone query, and relational operators that are sensitive to the semantic lattice structures of the domains. We also investigate the use of these operators in query relaxation. In Section 7 we show how clean answers can be approximated. In Section 8 we establish a connection to entity resolution as captured in generic terms by *Swoosh* [9]. In 9 we discuss ongoing and future research directions and some related issues; and also related work. We present concluding remarks in Section 10.

2 Background

A database schema $\mathcal{R}$ is a set $\{R_1, \ldots, R_n\}$ of relation names. Every relation R_i is associated with a set of attributes, written as $R_i(A_1, \ldots, A_m)$, where each attribute A_j has a domain Dom_{A_j}. We assume that attribute names are different across relations in the schema, but two attributes A_j, A_k can be *comparable*, i.e., $Dom_{A_j} = Dom_{A_k}$. An instance D of schema $\mathcal{R}$ assigns a finite set of tuples, each denoted by t, or t^D to emphasize its membership in D, to every relation R_i. Each t^D can be seen as a function that maps every attribute A_j in R_i to a value in Dom_{A_j}. We write $t^D[A_j]$ to refer to this value. The *active domain* of an attribute A for an instance D, denoted $adom(D, A)$, is the finite set that contains all the values for A from Dom_A that appear in D. Of course, for comparable attributes A_1, A_2 it may happen that $adom(D, A_1) \neq adom(D, A_2)$.

When X is a list of attributes, we may write $t^D[X]$ to refer to the corresponding list of attribute values. A tuple t^D for a relation name $R \in \mathcal{R}$ is called an R-tuple. We deal with queries $\mathcal{Q}$ that are expressed in relational algebra, and treat them as operators that map an instance D to an instance $\mathcal{Q}(D)$.

For every attribute A in the schema, we assume a binary similarity relation $\approx_A \subseteq Dom_A \times Dom_A$. Notice that whenever A and A' are comparable, the similarity relations $\approx_A$ and $\approx_{A'}$ are identical. We assume that each $\approx_A$ is reflexive and symmetric. When there is no confusion, we simply use $\approx$ for the similarity relation. In particular, for lists of pairwise comparable attributes, $X_i = A_1^i, \ldots, A_n^i$, $i = 1, 2$, we write $X_1 \approx X_2$ to mean $A_1^1 \approx_1 A_1^2 \wedge \cdots \wedge A_n^1 \approx_n A_n^2$, where $\approx_i$ is the similarity relation applicable to attributes A_i^1, A_i^2.

Given two pairs of pairwise comparable attribute lists X_1, X_2 and Y_1, Y_2 from relations R_1, R_2, resp., a *matching dependency* (MD) [20] is a sentence of the form

$$\varphi: \quad R_1[X_1] \approx R_2[X_2] \rightarrow R_1[Y_1] \rightleftharpoons R_2[Y_2].^1 \tag{1}$$

This dependency intuitively states that if for an R_1-tuple t_1 and an R_2-tuple t_2 in instance D, the attribute values in $t_1^D[X_1]$ are similar to attribute values in $t_2^D[X_2]$, then we need to make the values $t_1^D[Y_1]$ and $t_2^D[Y_2]$ pairwise identical.

Enforcing MDs may cause a database instance D to be changed to another instance D'. To keep track of every single change, we assume that every tuple t in an instance has a unique identifier, which we will also denote with t, and can be used to identify it in both instance D and its changed version D'. We can use the notations t^D and $t^{D'}$ introduced above to refer to a tuple in D and its changed version in D' that has resulted from enforcing an MD, resp. For convenience, we may use the terms tuple and tuple identifier interchangeably.

Fan et al. [20] give a *dynamic semantics* for matching dependencies in terms of a pair of instances: one where the similarities hold, and a second where the specified identifications have been enforced:

Definition 1. [20] A pair of instances (D, D') satisfies the MD $\varphi : R_1[X_1] \approx R_2[X_2] \rightarrow R_1[Y_1] \rightleftharpoons R_2[Y_2]$, denoted $(D, D') \models \varphi$, if for every R_1-tuple t_1 and R_2-tuple t_2 in D that match the left-hand side of φ, i.e., $t_1^D[X_1] \approx t_2^D[X_2]$, the following holds in the instance D':

(a) $t_1^{D'}[Y_1] = t_2^{D'}[Y_2]$, i.e., the values of the right-hand side attributes of φ have been identified in D'; and
(b) t_1, t_2 in D' match the left-hand side of φ, that is, $t_1^{D'}[X_1] \approx t_2^{D'}[X_2]$.

For a set Σ of MDs, $(D, D') \models \Sigma$ iff $(D, D') \models \varphi$ for every $\varphi \in \Sigma$. An instance D' is called *stable* if $(D', D') \models \Sigma$. ■

Notice that a stable instance satisfies the MDs by itself, in the sense that all the required identifications are already enforced in it. Whenever we say that an instance is *dirty*, we mean that it is not stable w.r.t. the given set of MDs.

While this definition may be sufficient for the implication problem of MDs considered by Fan et al. [20], it does not specify how a dirty database should be updated to obtain a clean instance, especially when several *interacting updates* are required in order to enforce all the MDs. Thus, it does not give a complete prescription for the purpose of cleaning dirty instances. Moreover, from a different perspective, the requirements in the definition may be too strong, as the following example shows.

Example 2. Consider the set of MDs Σ consisting of $\varphi_1 : R[A] \approx R[A] \rightarrow R[B] \rightleftharpoons R[B]$ and φ_2: $R[B, C] \approx R[B, C] \rightarrow R[D] \rightleftharpoons R[D]$. The similarities are: $a_1 \approx a_2$, $b_2 \approx b_3$, $c_2 \approx c_3$. Instance D_0 below is not a stable instance, i.e., it does not satisfy φ_1, φ_2. We start by enforcing φ_1 on D_0.

[1] All the variables in X_i, Y_j are implicitly universally quantified in front of the formula.

D_0	A	B	C	D
	a_1	b_1	c_1	d_1
	a_2	b_2	c_2	d_2
	a_3	b_3	c_3	d_3

D_1	A	B	C	D
	a_1	$\langle b_1, b_2\rangle$	c_1	d_1
	a_2	$\langle b_1, b_2\rangle$	c_2	d_2
	a_3	b_3	c_3	d_3

Let $\langle b_1, b_2\rangle$ in instance D_1 denote the value that replaces b_1 and b_2 to enforce φ_1 on instance D_0, and assume that $\langle b_1, b_2\rangle \not\approx b_3$. Now, $(D_0, D_1) \models \varphi_1$. However, $(D_0, D_1) \not\models \varphi_2$.

If we identify d_2, d_3 via $\langle d_2, d_3\rangle$ producing instance D_2, the pair (D_0, D_2) satisfies condition (a) in Definition 1 for φ_2, but not condition (b). Notice that making more updates on D_1 (or D_2) to obtain an instance D', such that $(D_0, D') \models \Sigma$, seems hopeless as φ_2 will not be satisfied because of the broken similarity that existed between b_2 and b_3. ■

Definition 1 seems to capture well the one-step enforcement of a single MD. However, as shown by the above example, the definition has to be refined in order to deal with sets of interacting MDs and to capture an iterative process of MD enforcement. We address this problem in Section 4.

Another issue worth mentioning is that stable instances D' for D and Σ are not subject to any form of minimality criterion on D' in relation with D. We would expect such an instance to be obtained via the enforcement of the MDs, without unnecessary changes. Unfortunately, this is not the case here: If in Example 2 we keep only φ_1, and in instance D_1 we change a_3 by an arbitrary value a_4 that is not similar to either a_1 or a_2, we obtain a stable instance with origin in D_0, but the change of a_3 is unjustified and unnecessary. We will also address this issue.

Following [20], we assume in the rest of this paper that each MD is of the form $R_1[X_1] \approx R_2[X_2] \rightarrow R_1[A_1] \rightleftharpoons R_2[A_2]$. That is, the right-hand side of each MD contains a pair of single attributes.

3 Matching Functions and Semantic Domination

In order to enforce a set of MDs (cf. Section 4) we need an operation that identifies two values whenever necessary. With this purpose in mind, we will assume that for each comparable pair of attributes A_1, A_2 with domain Dom_A, there is a binary *matching function* $m_A : Dom_A \times Dom_A \rightarrow Dom_A$, such that the value $m_A(a, a')$ is used to replace two values $a, a' \in Dom_A$ whenever the two values need to be made equal. Here are a few natural properties to expect of the matching function m_A (similar properties were considered in [9], cf. Section 8.2): For $a, a', a'' \in Dom_A$:

I (Idempotency): $m_A(a, a) = a$,
C (Commutativity): $m_A(a, a') = m_A(a', a)$,
A (Associativity): $m_A(a, m_A(a', a'')) = m_A(m_A(a, a'), a'')$.

It is reasonable to assume that any matching function satisfies at least these three axioms. Idempotency is a natural assumption as it is never desirable to replace two values that are already identical with a new value. Commutativity and associativity are also expected, intuitively because applying a matching function to make two or more values identical should not be sensitive to the order in which those values are visited. (We revisit the associativity property in Section 9.1.)

Under these assumptions, the structure (Dom_A, m_A) forms a *join semilattice*, $\mathcal{L}_A$, that is, a partial order with a least upper bound (*lub*) for every pair of elements. The induced partial order $\preceq_A$ on the elements of Dom_A is defined as follows: For every $a, a' \in Dom_A$, $a \preceq_A a'$ whenever $m_A(a, a') = a'$. The *lub* operator with respect to this partial order coincides with m_A: $lub_{\preceq_A}\{a, a'\} = m_A(a, a')$.

A natural interpretation for the partial order $\preceq_A$ in the context of data cleaning would be the notion of *semantic domination.* Basically, for two elements $a, a' \in Dom_A$, we say that a' *semantically dominates* a if we have $a \preceq_A a'$. In the process of cleaning the data by enforcing matching dependencies, we always replace two values a, a', whenever certain similarities hold, by the value $m_A(a, a')$ that semantically dominates both a and a'. This notion of domination is also related to relative information contents [13, 27, 29].

One of our key goals is to develop a semantic account of, and computational mechanisms for, obtaining clean instances from a given input instance using MDs together with matching functions. The assumptions about m_A mentioned above (which result in the existence of *lub* for every two elements in the domain) are enough for realizing this goal. However, it turns out we additionally need the existence of the greatest lower bound (*glb*) for any two elements in the domain of an attribute, in order to define the semantics of query answering on the clean instances obtained using MDs. In Section 5, we will make use of the existence *glb* to define and compute certain answers whenever there are multiple clean instances.

So far, we have assumed that the lattice-theoretic structure of an attribute domain Dom_A is created via a matching function m_A. However, it is also quite natural that, for an attribute A, its domain Dom_A comes already endowed with a lattice structure $\mathcal{L}_A = \langle Dom_A, \preceq_A \rangle$. As a consequence, for any two-element subset S of Dom_A, both $glb_{\preceq_A}(S)$ and $lub_{\preceq_A}(S)$ exist. We may also assume that $\mathcal{L}_A$ has *bottom* and *top* elements, generically denoted with $\perp, \top$, resp., such that $\perp \preceq_A a \preceq_A \top$ for every $a \in Dom_A$. On the basis of such a lattice structure on Dom_A, we could now *define* the matching function m_A by $m_A(a, b) := lub_{\preceq_A}\{a, b\}$. Of course, under this second alternative (i.e. using the lattice to define the matching function), for every $a \in Dom_A$, $m_A(a, \perp) = a$ and $m_A(a, \top) = \top$.

The presence of $\top$ allows us to have a *total* matching function m_A, because whenever two values, a, b, do not naturally match, we can set $m_A(a, b) := \top$. This element could represent the existence of inconsistency in data whenever the MDs force the matching of two completely unrelated elements from the domain. However, the existence of $\top$ is not essential in our framework.

Either way we go, i.e. starting from m_A or from a partial order $\preceq_A$, we will assume that (Dom_A, m_A) is a lattice (i.e., both *lub* and *glb* exist for every pair of elements in Dom_A w.r.t. $\preceq_A$). Notice that if we add an additional assumption to the semilattice, which requires that for every element $a \in Dom_A$, the set $\{c \in Dom_A \mid c \preceq_A a\}$ (the set of elements c with $m_A(a,c) = a$), is finite, then $glb_{\preceq_A}\{a, a'\}$ does exist for every two elements $a, a' \in Dom_A$ and is equal to $lub_{\preceq_A}\{c \in Dom_A \mid c \preceq_A a \text{ and } c \preceq_A a'\}$.

The choice and implementation of a matching function for each attribute domain is a decision that has to be made by a domain expert. A general matching function that could potentially work for every attribute domain is a function that treats attribute values as sets and takes the union of two sets whenever they need to be identified. For numerical domains, in an application, this can be followed by a step where an aggregation function such as average is applied. More specific matching functions could be used depending on the domain, as shown in the following example.

Example 3. We give a few concrete examples of matching functions for different attribute domains. Our example functions have all the properties **I**, **C**, and **A**.

(a) *Name, Address, Phone:* Each atomic value s of these string domains could be treated as a singleton set $\{s\}$. Then a matching function $m(S_1, S_2)$ for sets of strings S_i could return $S_1 \cup S_2$. E.g., when matching addresses, $m(\{$"2366 Main Mall"$\}, \{$"Main Mall, Vancouver"$\})$ could return the value $\{$"2366 Main Mall", "Main Mall, Vancouver"$\}$.
Alternatively, a more sophisticated matching function could merge two input strings into a third string that contains both of the inputs. E.g., the match of the two input strings above could instead be "2366 Main Mall, Vancouver". Part of the corresponding lattice is shown in Figure 1.

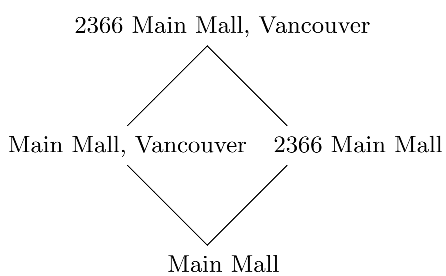


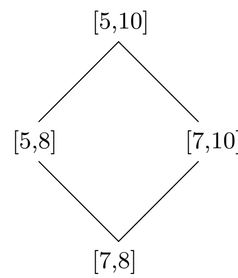
Fig. 1. A domain lattice

One way to formally reconstruct this kind of matching function is through the identification of an attribute value (actually, even whole records or tuples) with an *object*, in this case, a set of pairs (*Attribute Name*, *Value*) (with a common id). For example, the values "2366 Main Mall" and "Main Mall, Vancouver" are identified with the objects $\{(id, \mathit{House\ Number}, 2366), (id, \mathit{Street\ Name}, \text{Main Mall})\}$ and $\{(id, \mathit{Street\ Name}, \text{Main Mall}),$

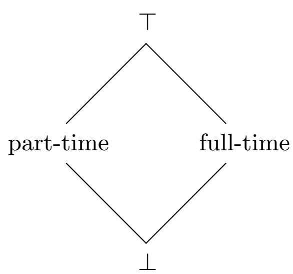
(id, *City*, Vancouver)}, respectively. If these values are matched through their union, we obtain {(id, *House Number*, 2366), (id, *Street Name*, Main Mall), (id, *City*, Vancouver)}, corresponding to "2366 Main Mall, Vancouver". The "union matching function" is further investigated in Section 8.2.

(b) *Age, Salary, Price:* Each atomic value a in these numerical domains could be treated as an interval $[a, a]$. Then the matching function $m([a_1, b_1], [a_2, b_2])$ would return the smallest interval containing both $[a_1, b_1]$ and $[a_2, b_2]$, i.e., $m([a_1, b_1], [a_2, b_2]) = [min\{a_1, a_2\}, max\{b_1, b_2\}]$.

An example of corresponding lattice is shown in the adjacent figure. In this case, the semantic domination is understood set-theoretically, specifically as interval inclusion.



(c) *Boolean Attributes:* For attributes which take either a 0 or 1 value, the matching function would return $m(0, 1) = \top$, where $\top$ shows inconsistency in the data, and furthermore $m(0, \top) = \top$ and $m(1, \top) = \top$.



In this case, the purpose of applying the matching function is to *record* the inconsistency in the data and still conduct sound reasoning in presence of inconsistency.[2] The adjacent figure shows an example of this kind of lattice. ■

An additional property of matching functions worthy of consideration is *similarity preservation*, that is, the result of applying a matching function preserves the similarity that existed between the old value being replaced and other values in the domain (a similar property was considered in [9], cf. Section 8.2). More formally, for every $a, a', a'' \in Dom_A$:

S (Similarity Preservation): If $a \approx a'$, then $a \approx m_A(a', a'')$.

Unlike the previous properties (**I**, **C**, and **A**), property **S** turns out to be a strong assumption, and we must consider both matching functions with **S** and without it. Indeed, notice that since $\approx$ is reflexive and m_A is commutative, assumption **S** implies $a \approx m_A(a, a')$ and $a' \approx m_A(a, a')$, i.e., similarity preserving matching always results in a value similar to the value being replaced. Actually, the following simple result will be used later on.

[2] Matching of boolean attributes requires the existence of the top element $\top$.

Proposition 1. Let m_A be a similarity preserving function, and a_1, a_2, a_3, a_4 be values in the domain Dom_A, such that $a_1 \preceq a_3$ and $a_2 \preceq a_4$. If $a_1 \approx_A a_2$, then $a_3 \approx_A a_4$. ■

In the rest of the paper, we assume that for every comparable pair of attributes A_1, A_2, there is an idempotent, commutative, and associative binary matching function m_A. Unless otherwise specified, we do not assume that these functions are similarity preserving.

Definition 2. Let D_1, D_2 be instances of schema $\mathcal{R}$, and t_1, t_2 be two R-tuples in D_1, D_2, respectively, with $R \in \mathcal{R}$. We write $t_1 \preceq t_2$ if $t_1^{D_1}[A] \preceq_A t_2^{D_2}[A]$ for every attribute A in R. We write $D_1 \sqsubseteq D_2$ if for every tuple t_1 in D_1, there is a tuple t_2 in D_2, such that $t_1 \preceq t_2$. ■

Clearly, the relation $\preceq$ on tuples can be applied to tuples in the same instance. The ordering $\sqsubseteq$ on sets has been used in the context of complex objects [8, 31] and also powerdomains, where it is called *Hoare ordering* [13]. It is also used in [9] for entity resolution (cf. Section 8.2). It is known that for $\sqsubseteq$ to be a partial order, specifically to be antisymmetric, we need to deal with *reduced* instances [8], i.e.,

Definition 3. For an instance D, its $\preceq$*-reduced version* is

$$Red_{\preceq}(D) = \{t \in D \mid \text{there is no tuple } t' \in D \text{ different from } t \text{ with } t \preceq t'\},$$

which is obtained from D by removing every tuple that is strictly dominated. ■

Next we will show that the set of reduced instances with the partial order $\sqsubseteq$ forms a lattice: the least upper bound and the greatest lower bound for every finite set of reduced instances exist. This result will be used later for query answering. We adapt some of the results from [8], where they prove a similar result for a lattice formed by the set of complex objects and the sub-object partial order.

Definition 4. Let D_1, D_2 be instances of schema $\mathcal{R}$, and t_1, t_2 be two R-tuples in D_1, D_2, respectively, for $R \in \mathcal{R}$.

(a) $D_1 \curlyvee D_2$ is $Red_{\preceq}(D_1 \cup D_2)$, where $D_1 \cup D_2$ is the set-theoretic union of D_1 and D_2.
(b) $t_1 \curlywedge t_2$ is the tuple t, such that $t[A] = glb_{\preceq_A}\{t_1^{D_1}[A], t_2^{D_2}[A]\}$ for every attribute A in R.
(c) $D_1 \curlywedge D_2$ is the instance that assigns, to each $R \in \mathcal{R}$, the set of tuples $Red_{\preceq}(\{t_1 \curlywedge t_2 \mid t_1 \in D_1,\ t_2 \in D_2, \text{and } t_1, t_2 \text{ are } R\text{-tuples}\})$. ■

Next we show that the operations defined in Definition 4 are equivalent to the greatest lower bound and least upper bound of instances w.r.t. the partial order $\sqsubseteq$.

Lemma 1. For every two instances D_1, D_2 and R-tuples t_1, t_2 in D_1, D_2, the following hold:

1. $D_1 \curlyvee D_2$ is the least upper bound of D_1, D_2 w.r.t. $\sqsubseteq$.
2. $t_1 \curlywedge t_2$ is the greatest lower bound of t_1, t_2 w.r.t. $\preceq$.
3. $D_1 \curlywedge D_2$ is the greatest lower bound of D_1, D_2 w.r.t. $\sqsubseteq$.

Proof: 1. Let D be the instance $D_1 \curlyvee D_2$. Clearly, $D_1 \sqsubseteq D$ and $D_2 \sqsubseteq D$. Now let D' be an arbitrary instance such that $D_1 \sqsubseteq D'$ and $D_2 \sqsubseteq D'$, and let t be a tuple in D. Then, by definition, t is in D_1 or in D_2, and hence there should be a tuple t' in D' such that $t^D \preceq t'^{D'}$. Therefore, we have $D \sqsubseteq D'$, and thus D is the least upper bound of D_1, D_2.
2. Let t be the tuple $t_1 \curlywedge t_2$. Clearly, $t \preceq t_1^{D_1}$ and $t \preceq t_2^{D_2}$. Let t' be an arbitrary tuple such that $t' \preceq t_1^{D_1}$ and $t' \preceq t_2^{D_2}$. Then $t'[A] \preceq t_1^{D_1}[A]$ and $t'[A] \preceq t_2^{D_2}[A]$ for every attribute A in the schema. Thus, $t'[A] \preceq glb(t_1^{D_1}[A], t_2^{D_2}[A])$ for every attribute A, and hence $t' \preceq t$.
3. Let D be the instance $D_1 \curlywedge D_2$. Let t be a tuple in D. Then there exist tuples t_1 in D_1 and t_2 in D_2, such that $t = t_1 \curlywedge t_2$, and thus $t^D \preceq t_1^{D_1}$ and $t^D \preceq t_2^{D_2}$. Therefore, it follows that $D \sqsubseteq D_1$ and $D \sqsubseteq D_2$.

Let D' be an arbitrary instance such that $D' \sqsubseteq D_1$ and $D' \sqsubseteq D_2$, and let t' be a tuple in D'. Then there exist tuples t_1 in D_1 and t_2 in D_2, such that $t'^{D'} \preceq t_1^{D_1}$ and $t'^{D'} \preceq t_2^{D_2}$, and thus $t'^{D'} \preceq glb(t_1^{D_1}, t_2^{D_2})$, which exists in D. We thus have $D' \sqsubseteq D$. ■

In particular, we can see that $\preceq$ imposes a lattice structure on R-tuples. Using Lemma 1, we immediately obtain the following result.

Theorem 1. The set of reduced instances for a given schema with the $\sqsubseteq$ ordering forms a lattice. ■

Example 4. Consider the following instances

D'	*name*	*phone*	*address*
	John Doe	(613)123 4567	25 Main St., Ottawa
	J. Doe	(613)123 4567	25 Main St., Ottawa
	Jane Doe	(604)123 4567	25 Main St., Vancouver

D''	*name*	*phone*	*address*
	John Doe	(613)123 4567	Main St., Ottawa
	J. Doe	(604)123 4567	25 Main St., Vancouver
	Jane Doe	(604)123 4567	25 Main St., Vancouver

The domain of three attributes involved conform to the lattice structures shown in Figure 2. They are of the kind shown in Figure 1, i.e., the *lub* or *m* of two string values is a string that merges them whenever it makes sense. Notice that an alternative lattice would be the subset lattice, when the *lub* of two string sets is the union of the two sets.

The instance $\{t' \curlywedge t'' \mid t' \in D',\ t'' \in D'', t_1, t_2\}$ is:

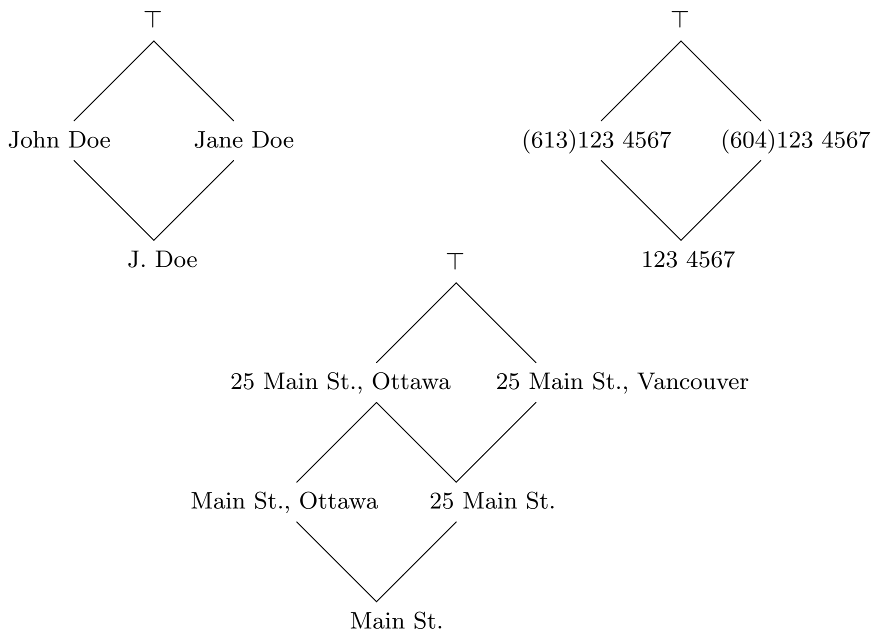


Fig. 2. Domain lattices $\mathcal{L}_{name}$, $\mathcal{L}_{phone}$, and $\mathcal{L}_{address}$

name	*phone*	*address*
John Doe	(613)123 4567	Main St., Ottawa
J. Doe	123 4567	25 Main St.
J. Doe	123 4567	25 Main St.
J. Doe	(613)123 4567	Main St., Ottawa
J. Doe	123 4567	25 Main St.
J. Doe	123 4567	25 Main St.
J. Doe	123 4567	Main St.
J. Doe	(604)123 4567	25 Main St., Vancouver
Jane Doe	(604)123 4567	25 Main St., Vancouver

After reduction, we obtain

$D' \curlywedge D''$	*name*	*phone*	*address*
	John Doe	(613)123 4567	Main St., Ottawa
	Jane Doe	(604)123 4567	25 Main St., Vancouver

which is the $glb_{\sqsubseteq}(D', D'')$. ■

4 Enforcement of MDs and Clean Instances

In this section, we define *clean* instances that can be obtained from a dirty instance by iteratively enforcing a set of MDs in a chase-like procedure.

Definition 5. Let D, D' be database instances with the same set of tuple identifiers, Σ be a set of MDs, and $\varphi : R_1[X_1] \approx R_2[X_2] \rightarrow R_1[A_1] \rightleftharpoons R_2[A_2]$ be an MD in Σ. Let t_1, t_2 be an R_1-tuple and an R_2-tuple identifiers, respectively, in both D and D'. We say that instance D' is the *immediate result of enforcing* φ on t_1, t_2 in instance D, denoted $(D, D')_{[t_1,t_2]} \models \varphi$, if

(a) $t_1^D[X_1] \approx t_2^D[X_2]$, but $t_1^D[A_1] \neq t_2^D[A_2]$;
(b) $t_1^{D'}[A_1] = t_2^{D'}[A_2] = m_A(t_1^D[A_1], t_2^D[A_2])$; and
(c) D, D' agree on every other tuple and attribute value. ■

Definition 5 captures a single step in a chase-like procedure that starts from a dirty instance D_0 and enforces MDs step by step, by applying matching functions, until the instance becomes stable. We propose that the output of this chase should be treated as a clean version of the original instance for a given a set of MDs. This is formally defined as follows.

Definition 6. For an instance D_0 and a set of MDs Σ, an instance D_k is (D_0, Σ)-*clean* if D_k is stable, and there exists a finite sequence of instances $D_1, \ldots, D_{k-1}$ such that, for every $i \in [1, k]$, $(D_{i-1}, D_i)_{[t_1^i, t_2^i]} \models \varphi$, for some $\varphi \in \Sigma$ and tuple identifiers t_1^i, t_2^i. We write *Clean*(D_0, Σ) to denote the set of (D_0, Σ)-clean instances of D_0 w.r.t. Σ. ■

Notice that if $(D_0, D_0) \models \Sigma$, i.e., it is already stable, then D_0 is its only (D_0, Σ)-clean instance. Moreover, we have $D_{i-1} \sqsubseteq D_i$, for every $i \in [1, k]$, since we are using matching functions to identify values, and the application of matching functions leads to instances that semantically dominate the instances they replace. In particular, we have $D_0 \sqsubseteq D_k$. In other words, clean instance D_k semantically dominates dirty instance D_0, and we might say D_k it is *more informative* than D_0 in the sense that every tuple has been replaced by a newer version of the tuple that contains a more complete piece of information.

Theorem 2. Let Σ be a set of matching dependencies and D_0 be an instance. Then every sequence $D_1, D_2, \ldots$ such that, for every $i \geq 1$, $(D_{i-1}, D_i)_{[t_1^i, t_2^i]} \models \varphi$, for some $\varphi \in \Sigma$ and tuple identifiers t_1^i, t_2^i in D_{i-1}, is finite and computes a (D_0, Σ)-clean instance D_k in polynomial number of steps in the size of D_0. It also holds that $D_0 \sqsubseteq D_1 \sqsubseteq \ldots \sqsubseteq D_k$.

Proof: Suppose an MD $\varphi : R_1[X_1] \approx R_2[X_2] \rightarrow R_1[A_1] \rightleftharpoons R_2[A_2]$ is enforced on a pair of tuples $t_1^{D_{i-1}}, t_2^{D_{i-1}} \in D_{i-1}$ and let $t_1^{D_i}, t_2^{D_i}$ be the corresponding tuples in D_i after the MD is enforced. It is easy to see that at least one of the following must hold: $t_1^{D_{i-1}}[A_1] \preceq_A t_1^{D_i}[A_1]$ but $t_1^{D_{i-1}}[A_1] \neq t_1^{D_i}[A_1]$, OR $t_2^{D_{i-1}}[A_2] \preceq_A t_2^{D_i}[A_2]$ but $t_2^{D_{i-1}}[A_2] \neq t_2^{D_i}[A_2]$. That is, at least one of $t_1^{D_{i-1}}[A_1], t_2^{D_{i-1}}[A_2]$ must strictly grow w.r.t. the partial order $\preceq_A$. In other words, after each MD application, at least one tuple changes and the change is a growth w.r.t. a partial order $\preceq_A$ on one of its attributes. Now, consider the instance D_{max} consisting of exactly one tuple in every relation, for which the value of every attribute A is $lub_{\preceq}\{a \mid a \in adom(D_0, B)$ attribute B is A or comparable to $A\}$.[3] Clearly,

[3] Remember that comparable attributes share the similarity relation and matching function (and lattice structure).

D_{max} is an upper bound on every instance in any chase sequence. Moreover, the number of matching function applications required to produce each tuple in D_{max} is polynomial in the size of the original instance D_0. This is because for every attribute A, any arbitrary sequence of applying the matching function m_A on all the values appearing in the active domain of A (and its comparable attributes) would result in computing the required $lub_{\preceq}$ mentioned above.

From this, it follows that the stable instance D_k associated with every chase sequence can be obtained in a finite number of steps which is polynomial in the size of D_0. ■

This result says that the sequence of instances obtained by chasing MDs reaches a fixpoint after polynomial number of steps, which is guaranteed to be a stable instance w.r.t. all MDs. This is the consequence of assuming that matching functions are idempotent, commutative, and associative.

Observe that, for a given instance D_0 and set of MDs Σ, many clean instances may exist, each resulting from a different order of applications of MDs on D_0 and from different selections of violating tuples. The number of possible clean instances is clearly finite.

Notice also that for a (D_0, Σ)-clean instance D_k, we may have $(D_0, D_k) \not\models \Sigma$ (cf. Definition 1). Intuitively, the reason is that some of the similarities that existed in D_0 could have been broken by iteratively enforcing the MDs to produce D_k. We argue that this is a price we may have to pay if we want to enforce a set of interacting MDs. However, each (D_0, Σ)-clean instance is stable and captures the persistence of attribute values that are not affected by MDs.

The following example illustrates these points. We simply write $\langle a_1, \ldots, a_l \rangle$ instead of $m_A(a_1, m_A(a_2, m_A(\ldots, a_l)))$. This notation is well defined by virtue of the associativity assumption.

Example 5. Consider the set of MDs Σ consisting of $\varphi_1: R[A] \approx R[A] \rightarrow R[B] \rightleftharpoons R[B]$ and φ_2: $R[B] \approx R[B] \rightarrow R[C] \rightleftharpoons R[C]$. We have the similarities: $a_1 \approx a_2$, $b_2 \approx b_3$. The following sequence of instances leads to a (D_0, Σ)-clean instance D_2:

D_0	A	B	C
	a_1	b_1	c_1
	a_2	b_2	c_2
	a_3	b_3	c_3

D_1	A	B	C
	a_1	$\langle b_1, b_2 \rangle$	c_1
	a_2	$\langle b_1, b_2 \rangle$	c_2
	a_3	b_3	c_3

D_2	A	B	C
	a_1	$\langle b_1, b_2 \rangle$	$\langle c_1, c_2 \rangle$
	a_2	$\langle b_1, b_2 \rangle$	$\langle c_1, c_2 \rangle$
	a_3	b_3	c_3

However, $(D_0, D_2) \not\models \Sigma$, and the reason is that $\langle b_1, b_2 \rangle \approx b_3$ does not necessarily hold. We can enforce the MDs in another order and obtain a different (D_0, Σ)-clean instance:

D_0	A	B	C
	a_1	b_1	c_1
	a_2	b_2	c_2
	a_3	b_3	c_3

D_1'	A	B	C
	a_1	b_1	c_1
	a_2	b_2	$\langle c_2, c_3 \rangle$
	a_3	b_3	$\langle c_2, c_3 \rangle$

D_2'	A	B	C
	a_1	$\langle b_1, b_2 \rangle$	c_1
	a_2	$\langle b_1, b_2 \rangle$	$\langle c_2, c_3 \rangle$
	a_3	b_3	$\langle c_2, c_3 \rangle$

D_3'	A	B	C
	a_1	$\langle b_1, b_2 \rangle$	$\langle c_1, c_2, c_3 \rangle$
	a_2	$\langle b_1, b_2 \rangle$	$\langle c_1, c_2, c_3 \rangle$
	a_3	b_3	$\langle c_2, c_3 \rangle$

Again, D_3' is a (D_0, Σ)-clean instance, but $(D_0, D_3') \not\models \Sigma$. ■

It would be interesting to know when there is only one (D_0, Σ)-clean instance D_k, and also when, for a clean instance D_k, $(D_0, D_k) \models \Sigma$ holds. The following two propositions establish natural sufficient conditions for these properties to hold.

Proposition 2. If every matching function m_A is similarity preserving, then, for every set of MDs Σ and every instance D_0, there is a unique (D_0, Σ)-clean instance D. Furthermore, $(D_0, D) \models \Sigma$. ∎

For the proof we state first the following lemma.

Lemma 2. Assume the matching functions are similarity preserving. Let $D_1, \dots,$ D_k be a sequence of instances such that D_k is stable, and for every $i \in [1, k]$, $(D_{i-1}, D_i)_{[t_1^i, t_2^i]} \models \varphi$, for some $\varphi \in \Sigma$ and tuple identifiers t_1^i, t_2^i. Let D be a (D_0, Σ)-clean instance not necessarily equal to D_k. Then for every $i \in [0, k]$, the following holds:

1. $t^{D_i}[A_1] \preceq t^D[A_1]$, for every tuple identifier t and every attribute A_1.
2. if $t^{D_i}[A_1] \approx t'^{D_i}[A_2]$, then $t^D[A_1] \approx t'^D[A_2]$, for every two tuple identifiers t, t' and two comparable attributes A_1, A_2.

Proof: The proof of this lemma is by an induction on i. For $i = 0$, we clearly have $t^{D_0}[A_1] \preceq t^D[A_1]$ since D is a (D_0, Σ)-clean instance. Moreover, if $t^{D_0}[A_1] \approx t'^{D_0}[A_2]$, then $t^D[A_1] \approx t'^D[A_2]$ by Proposition 1.

Suppose 1. and 2. hold for every $i < j$. If 1. holds for $i = j$, then 2. also holds for $i = j$ by Proposition 1. Suppose 1. does not hold for $i = j$: $t^{D_j}[A_1] \not\preceq t^D[A_1]$. Since 1. holds for every $i < j$, the value of $t^{D_j}[A_1]$ should be different from $t^{D_{j-1}}[A_1]$. Therefore, there should be an MD $\varphi : R_1[X_1] \approx R_2[X_2] \rightarrow R_1[A_1] \rightleftharpoons R_2[A_2]$ in Σ and a tuple identifier t', such that D_j is the immediate result of enforcing φ on t, t' in D_{j-1}. That is, $t^{D_{j-1}}[X_1] \approx t'^{D_{j-1}}[X_2]$, $t^{D_{j-1}}[A_1] \neq t'^{D_{j-1}}[A_2]$, and $t^{D_j}[A_1] = t'^{D_j}[A_2] = m_A(t^{D_{j-1}}[A_1], t'^{D_{j-1}}[A_2])$. Since $t^{D_{j-1}}[X_1] \approx t'^{D_{j-1}}[X_2]$, by induction hypothesis, we have $t^D[X_1] \approx t'^D[X_2]$, and thus, $t^D[A_1] = t'^D[A_2]$, because D is a stable instance. Again by induction hypothesis, $t^{D_{j-1}}[A_1] \preceq t^D[A_1]$ and $t'^{D_{j-1}}[A_2] \preceq t'^D[A_2] = t^D[A_1]$. Therefore, $t^{D_j}[A_1] = m_A(t^{D_{j-1}}[A_1], t'^{D_{j-1}}[A_2]) \preceq t^D[A_1]$ since m_A takes the least upper bound, which leads to a contradiction. ∎

Proof of Proposition 2: Let D, D' be two (D_0, Σ)-clean instances. To prove the first part of the proposition, notice that, from Lemma 2, we obtain $t^D[A] \preceq t^{D'}[A]$ and $t^{D'}[A] \preceq t^D[A]$ for every tuple identifier t and every attribute A. Thus, the two (D_0, Σ)-clean instances D, D' should be identical.

To prove the second part of the proposition, let $\varphi : R_1[X_1] \approx R_2[X_2] \rightarrow R_1[A_1] \rightleftharpoons R_2[A_2]$ be an MD in Σ, and let D be the unique (D_0, Σ)-clean instance. By Lemma 2, if $t_1^{D_0}[X_1] \approx t_2^{D_0}[X_2]$, then $t_1^D[X_1] \approx t_2^D[X_2]$, for every two tuple identifiers t_1, t_2. Since D is a stable instance, $t_1^D[A_1] = t_2^D[A_2]$, and thus $(D_0, D) \models \varphi$ and $(D_0, D) \models \Sigma$. ∎

Definition 7. A set of MDs Σ is *interaction-free* if for every two MDs $\varphi_1, \varphi_2 \in \Sigma$, not necessarily distinct, the set of attributes on the right-hand side of φ_1 is disjoint from the set of attributes on the left-hand side of φ_2. $\blacksquare$

Notice that the two sets of MDs in Examples 2 and 5 are not interaction-free.

Proposition 3. Let Σ be an interaction-free set of MDs. Then, for every instance D_0, there is a unique (D_0, Σ)-clean instance D. Furthermore, $(D_0, D) \models \Sigma$. $\blacksquare$

The proof of this proposition immediately follows from the following lemma, which is similar to Lemma 2.

Lemma 3. Let Σ be an interaction-free set of MDs. Also let $D_1, \ldots, D_k$ be a sequence of instances such that D_k is stable, and for every $i \in [1, k]$, $(D_{i-1}, D_i)_{[t_1^i, t_2^i]} \models \varphi$, for some $\varphi \in \Sigma$ and tuple identifiers t_1^i, t_2^i. Let D be a (D_0, Σ)-clean instance not necessarily equal to D_k. Then for every $i \in [0, k]$, the following holds:

1. For every two tuple identifiers t, t' and every MD $\varphi \in \Sigma$, $t^{D_i}[X_1] = t^{D_0}[X_1]$ and $t'^{D_i}[X_2] = t'^{D_0}[X_2]$, where X_1, X_2 are the lists of attributes on the left-hand side of φ.
2. $t^{D_i}[A] \preceq t^D[A]$, for every tuple identifier t and every attribute A.

Proof: Notice that 1. trivially holds: since MDs are interaction-free, there is no MD $\varphi' \in \Sigma$, such that the attributes on the right-hand side of φ' has an intersection with X_1, X_2, and therefore no MD enforcement could change the values in $t^{D_i}[X_1]$ or $t'^{D_i}[X_2]$ into something different from the original values in D_0.

We prove 2. by an induction on i. For $i = 0$, we clearly have $t^{D_0}[A] \preceq t^D[A]$ since D is a (D_0, Σ)-clean instance. Now suppose 2. holds for $i < j$, and it does not hold for $i = j$: $t^{D_j}[A] \npreceq t^D[A]$. Then there should be an MD $\varphi : R_1[X_1] \approx R_2[X_2] \rightarrow R_1[A] \rightleftharpoons R_2[A']$ in Σ and a tuple identifier t', such that D_j is the immediate result of enforcing φ on t, t' in D_{j-1}. That is, $t^{D_{j-1}}[X_1] \approx t'^{D_{j-1}}[X_2]$, $t^{D_{j-1}}[A] \neq t'^{D_{j-1}}[A']$, and $t^{D_j}[A] = t'^{D_j}[A'] = m_A(t^{D_{j-1}}[A], t'^{D_{j-1}}[A'])$. Since $t^{D_{j-1}}[X_1] \approx t'^{D_{j-1}}[X_2]$, by part 1 we have $t^D[X_1] \approx t'^D[X_2]$, and thus $t^D[A] = t'^D[A']$, because D is a stable instance. By induction assumption, $t^{D_{j-1}}[A] \preceq t^D[A]$ and $t'^{D_{j-1}}[A'] \preceq t'^D[A'] = t^D[A]$. Therefore, $t^{D_j}[A] = m_A(t^{D_{j-1}}[A], t'^{D_{j-1}}[A']) \preceq t^D[A]$, since m_A takes the least upper bound, which leads to a contradiction. $\blacksquare$

The chase-like procedure that produces a (D_0, Σ)-clean instance makes only those changes to instance D_0 that are necessary, and are imposed by the dynamic semantics of MDs. In this sense, we can say that the chase implements minimal changes necessary to obtain a clean instance.

Another interesting question is whether (D_0, Σ)-clean instances are at a minimal distance to D_0 w.r.t. the partial order $\sqsubseteq$. This is not true in general. For instance in Example 5, observe that for the two (D_0, Σ)-clean instances D_2 and

D_3', $D_2 \sqsubseteq D_3'$, but $D_3' \not\sqsubseteq D_2$, which means D_3' is not at a minimal distance to D_0 w.r.t. $\sqsubseteq$. We have actually no reason to expect the clean instances to be minimal in this sense since they are obtained as fixpoints of two different chase paths. However, both of these clean instances may be useful in query answering, because, informally speaking, they can provide a lower bound and an upper bound for the possible clean interpretations of the original dirty instance w.r.t. the semantic domination. This issue is discussed in the next section.

5 Clean Query Answering

Most of the literature on data cleaning has concentrated on producing a clean instance starting from a dirty one. However, the problem of characterizing and retrieving the data in the original instance that can be considered to be clean has been neglected. In this section we study this problem, focusing on query answering. More precisely, given an instance D, a set Σ of MDs, and a query $\mathcal{Q}$ posed to D, we want to characterize the answers that are consistent with Σ, i.e., that would be returned by an instance where all the MDs have been enforced. Of course, we have to take into account that there may be several such instances.

This situation is similar to the one encountered in *consistent query answering* (CQA) [5, 10, 15, 11], where query answering is characterized and performed on database instances that may fail to satisfy certain classic integrity constraints (ICs). For such a database instance, a *repair* is an instance that satisfies the integrity constraints and minimally differs from the original instance. For a given query, a *consistent answer* (a form of certain answer) is defined as the set of tuples that are present in the intersection of answers to the query when posed to every repair. A less popular alternative is the notion of *possible answer*, that is defined as the union of all tuples that are present in the answer to the query when posed to every repair.

A similar semantics for clean query answering under matching dependencies can be defined. However, the partial order relationship $\sqsubseteq$ between a dirty instance and its clean instances establishes an important difference between clean instances w.r.t. matching dependencies and repairs w.r.t. traditional ICs.

Intuitively, a clean instance has improved the information that already existed in the dirty instance and made it more informative and consistent. We would like to carefully take advantage of this partial order relationship and use it in the definition of certain and possible answers. We do this by taking the greatest lower bound (glb) and least upper bound (lub) of answers of the query over multiple clean instances, instead of taking the set-theoretic intersection [29] and union.

Definition 8. Let Σ be a set of MDs, D_0 be a database instance, and $\mathcal{Q}$ be a query. The *certain* and *possible answers* to $\mathcal{Q}$ from D_0 are defined as follows:

$$Cert_{\mathcal{Q}}(D_0) = glb_{\sqsubseteq}\{\mathcal{Q}(D) \mid D \text{ is a } (D_0, \Sigma)\text{-clean instance}\}, \quad (2)$$

$$Poss_{\mathcal{Q}}(D_0) = lub_{\sqsubseteq}\{\mathcal{Q}(D) \mid D \text{ is a } (D_0, \Sigma)\text{-clean instance}\}, \quad (3)$$

respectively. ■

The glb and lub above are defined on the basis of the partial order $\sqsubseteq$ on sets of tuples. Since there is a finite number of clean instances for D_0, these glb and lub exist (cf. Theorem 1). In Eq. (2) and (3) we are assuming that each $\mathcal{Q}(D)$ is reduced (cf. Section 3). By Definition 4 and Lemma 1, $Cert_{\mathcal{Q}}(D_0)$ and $Poss_{\mathcal{Q}}(D_0)$ are also reduced. Moreover, we clearly have $Cert_{\mathcal{Q}}(D_0) \sqsubseteq Poss_{\mathcal{Q}}(D_0)$.

Remark 1. If the query in Definition 8 is boolean, i.e. a sentence, then, for an instance D, $\mathcal{Q}(D) := \{yes\}$ when $\mathcal{Q}$ is true in D, and $\{no\}$, otherwise. We also assume that $no \preceq yes$, but $yes \not\preceq no$, creating a two-valued lattice. Accordingly, we define $\{no\} \sqsubseteq \{yes\}$. With this definition we can also give a natural account of boolean queries: $Cert_{\mathcal{Q}}(D_0) = \{yes\}$ iff $\mathcal{Q}(D) = \{yes\}$ for every (D_0, Σ)-clean instance D. Similarly, $Poss_{\mathcal{Q}}(D_0) = \{yes\}$ iff $\mathcal{Q}(D) = \{yes\}$ for some (D_0, Σ)-clean instance D.[4] ■

The following example motivates these choices. It also shows that, unlike some cases of inconsistent databases and consistent query answering [10], certain answers could be quite informative and meaningful for databases with matching dependencies.

Example 6. Consider relation $R(name, phone, address)$, and set Σ consisting of the following MDs:

φ_1: $R[name, phone, address] \approx R[name, phone, address] \rightarrow R[address] \rightleftharpoons R[address]$
φ_2: $R[phone, address] \approx R[phone, address] \rightarrow R[phone] \rightleftharpoons R[phone]$.

Suppose that in the dirty instance D_0, shown below, the following similarities hold:

"John Doe" $\approx$ "J. Doe", "Jane Doe" $\approx$ "J. Doe",
"(613)123 4567" $\approx$ "123 4567", "(604)123 4567" $\approx$ "123 4567",
"25 Main St." $\approx$ "Main St., Ottawa", "25 Main St." $\approx$ "25 Main St., Vancouver".

Other non-trivial similarities that are not mentioned do not hold. Moreover, the matching functions act as follows:

m_{phone}("(613)123 4567", "123 4567") = "(613)123 4567",
m_{phone}("123 4567", "(604)123 4567") = "(604)123 4567",
$m_{address}$("Main St., Ottawa", "25 Main St.") = "25 Main St., Ottawa",
$m_{address}$("25 Main St.", "25 Main St., Vancouver") = "25 Main St., Vancouver".

Notice that these values are consistent with (or better, emerge from) the lattices $\mathcal{L}_{phone}, \mathcal{L}_{address}$ (implicitly) introduced in Example 4; in the sense that $m_A(a, b) = lub_{\mathcal{L}_A}\{a, b\}$. It is also the case that $m_A(a, glb_{\mathcal{L}_A}\{a, b\}) = a$. Furthermore, notice that, for example, $m_{address}$("25 Main St., Ottawa", "25 Main St., Vancouver") = $\top$. In Example 4 we also have a lattice $\mathcal{L}_{name}$, for the *Name* attribute, even without having an explicit matching function for it or MDs that involve it in the right-hand side. We can still use $\mathcal{L}_{name}$ for establishing semantic domination between attributes values, tuples, and instances.

[4] For the same purpose we could also use the classic four-value lattice: $\bot \preceq false, true \preceq \top$, like the one in Example 3(c).

D_0	*name*	*phone*	*address*
	John Doe	(613)123 4567	Main St., Ottawa
	J. Doe	123 4567	25 Main St.
	Jane Doe	(604)123 4567	25 Main St., Vancouver

Observe that from D_0 we can obtain two different (D_0, Σ)-clean instances D', D'', depending on the order of enforcing MDs on different pairs of tuples.

D'	*name*	*phone*	*address*
	John Doe	(613)123 4567	25 Main St., Ottawa
	J. Doe	(613)123 4567	25 Main St., Ottawa
	Jane Doe	(604)123 4567	25 Main St., Vancouver

D''	*name*	*phone*	*address*
	John Doe	(613)123 4567	Main St., Ottawa
	J. Doe	(604)123 4567	25 Main St., Vancouver
	Jane Doe	(604)123 4567	25 Main St., Vancouver

Notice that these are the instances D', D'' in Example 4.

Now consider the query $\mathcal{Q} : \pi_{address}(\sigma_{name=\text{“J. Doe”}} R)$, asking for the residential address of J. Doe. We are interested in a certain answer. It can be obtained by taking the greatest lower bound of the two answer sets:

$$\begin{aligned}\mathcal{Q}(D') &= \{(\text{“25 Main St., Ottawa”})\},\\ \mathcal{Q}(D'') &= \{(\text{“25 Main St., Vancouver”})\}.\end{aligned}$$

In this case, and according to [8], and using Lemma 1,

$$\begin{aligned}glb_{\sqsubseteq}\{\mathcal{Q}(D'), \mathcal{Q}(D'')\} &= \{a' \curlywedge a'' \mid a' \in \mathcal{Q}(D'),\ a'' \in \mathcal{Q}(D'')\}\\ &= \{(\text{“25 Main St., Ottawa”}) \curlywedge (\text{“25 Main St., Vancouver”})\}\\ &= \{(glb_{\preceq_{address}}\{\text{“25 Main St., Ottawa”}, \text{“25 Main St., Vancouver”}\})\}\\ &= \{(\text{“25 Main St.”})\}.\end{aligned}$$

We can see that, no matter how we clean D_0, we can say for sure that J. Doe is at 25 Main St. Notice that the set-theoretic intersection of the two answer sets is empty. If we were interested in all possible answers, we could take the least upper bound of two answer sets, which would be the union of the two in this case. ■

We define a *clean answer* to be a pair consisting of an upper and lower bound of query answers over all possible clean interpretations of a dirty database instance. This definition is inspired by the same kind of approximations used in the contexts of partial and incomplete information [36, 1], inconsistent databases [5, 10, 15], and data exchange [35]. These upper and lower bounds could provide useful information about the value of aggregate functions, such as *sum* and *count* [7, 22, 3].

Remark 2. Considering Definition 8, and the fact that for a query $\mathcal{Q}$ posed to a database instance D_0 and a set of MDs Σ, $Cert_{\mathcal{Q}}(D_0) \sqsubseteq Poss_{\mathcal{Q}}(D_0)$, we can say that the *clean answers* to $\mathcal{Q}$ are specified by the two bounds or, equivalently, by the "interval" $\langle Cert_{\mathcal{Q}}(D_0), Poss_{\mathcal{Q}}(D_0)\rangle$. Notice that in the case of similarity-preserving matching functions or non-inter-acting matching dependencies, from the results in Section 4, these bounds would collapse into a single set, which is obtained by running the query on the unique clean instance. ■

5.1 Complexity of Computing Clean Answers

Here we study the complexity of computing clean answers over database instances in presence of MDs. As with incomplete and inconsistent databases, this problem easily becomes intractable for simple MDs and queries, which motivates the need for developing approximate solutions to the problem. We explore approximate solutions for queries that behave monotonically w.r.t. the partial order $\sqsubseteq$ in Section 7.

Theorem 3. (*complexity of clean query answering*) There are a schema with two interacting MDs and a relational algebra query, for which deciding whether a tuple belongs to the certain answer set for an instance D_0 is *coNP*-complete (in the size of D_0).

Proof: Consider relation schema $R(C, V, L)$, the conjunctive boolean query $\mathcal{Q}: \pi_L(R)(\top)$, and set Σ consisting of two MDs $\varphi_1 : R[C] \approx R[C] \rightarrow R[C] \rightleftharpoons R[C]$ and $\varphi_2 : R[CV] \approx R[CV] \rightarrow R[L] \rightleftharpoons R[L]$. The domains of attributes, similarity relations, and matching functions are as follows: $Dom_C = \{\bot, c, c_1, d_1, c_2, d_2, \ldots\}$, $Dom_V = \{\bot, y, x_1, x_2, \ldots\}$, $Dom_L = \{\bot, \top, +, -\}$. For every $c_i, d_i \in Dom_C$, we have $c_i \approx d_i$ and $m_C(c_i, d_i) = m_C(d_i, c_i) = c$. We also have $m_L(+, -) = m_L(-, +) = \top$. Notice that similarity relations and match functions are not fully described here. The full descriptions can be derived using the reflexivity and symmetry of similarity relations and idempotency, commutativity, and associativity of match functions.

In this case, we confront the problem of deciding membership of $\mathcal{C} := \{D_0 \mid Cert_{\mathcal{Q}}(D_0) = \{yes\}\}$. Its complement is $\mathcal{C}^c = \{D_0 \mid Cert_{\mathcal{Q}}(D_0) = \{no\}\}$. An instance D_0 belongs to $\mathcal{C}^c$ iff there is a *cleaning chase history* h that, starting at D_0, produces a clean instance D that makes $\mathcal{Q}$ false. Such a history h describes the sequence of applications of MDs starting from the initial instance, and it also includes for each of them the pair of tuples to which it was applied.

A non-deterministic algorithm to do this checking consists of guessing such a history h, and checking that: (a) it is applied according to the chase rules, (b) it leads to a stable instance D, and (c) D makes $\mathcal{Q}$ false. Notice that when such a *certificate* h exists, its size is polynomial in the size of D_0, and (a)-(c) can be all verified in polynomial time.

To prove hardness, we reduce from 3SAT. Let $\mathbf{C} = C_1 \wedge \ldots \wedge C_N$ be CNF formula, where each clause C_i, $i \in [1, N]$, is a disjunction of three literals $l_{i1} \vee l_{i2} \vee l_{i3}$, and each literal l_{ik}, $k \in [1, 3]$, is either x_j or $\neg x_j$ for some variable

$x_j \in Dom_V$. We create an instance D_0 of R as follows. For every clause C_i and every literal l_{ik} of variable x_j in C_i, there is a tuple t with $t^{D_0}[C] = c_i$, $t^{D_0}[V] = x_j$, $t^{D_0}[L] = +$ if $l_{ik} = x_j$ (a positive literal), and $t^{D_0}[L] = -$ if $l_{ik} = \neg x_j$ (a negative literal). Moreover, for every clause C_i, there is another tuple t with $t^{D_0}[C] = d_i$, $t^{D_0}[V] = y$, and $t^{D_0}[L] = +$.

We show that the CNF formula $\mathbf{C}$ is satisfiable if and only if $Cert_{\mathcal{Q}}(D_0) = \{no\}$. Let $\mathbf{C}$ be a satisfiable formula. For each clause C_i, we pick a tuple corresponding to the literal that is made true in the satisfying assignment and also the only tuple with $t^{D_0}[C] = d_i$, and enforce the MD φ_1 on these two tuples. It is easy to see that the result would be a stable instance D. In particular, $(D, D) \models \varphi_2$ since for each variable the satisfying assignment has picked only one of the positive or negative literals to be true. Therefore, we do not need to enforce φ_2, which means that $\top$ does not appear for any value of attribute L, and hence $\mathcal{Q}(D) = \{no\}$ and $Cert_{\mathcal{Q}}(D_0) = \{no\}$.

Conversely, if $Cert_{\mathcal{Q}}(D_0) = \{no\}$, there is a (D_0, Σ)-clean instance D in which $\top$ does not appear for any value of attribute L. To obtain the clean instance D starting from D_0, we need to enforce φ_1 once for each clause C_i, as described above, before we can enforce φ_2 on any tuple corresponding to C_i. Moreover, for every two tuples in D that match the left-hand side of φ_2, we should have identical values for attribute L (either $+$ or $-$), otherwise we would get $\top$ when enforcing φ_2. Therefore, for each clause, we can make true the literal corresponding to the tuple on which φ_1 has been enforced, and obtain a correct satisfying assignment. ■

6 Monotone Queries

So far we have seen that clean instances are a more informative view of a dirty instance obtained by enforcing matching dependencies. That is, $D_0 \sqsubseteq D$, for every (D_0, Σ)-clean instance D. From this perspective, it would be natural to expect that for a positive query, we would obtain a more informative answer if we pose it to a clean instance instead of to the dirty one. We can translate this requirement into a monotonicity property for queries w.r.t. the partial order $\sqsubseteq$.

Definition 9. A query $\mathcal{Q}$ is $\sqsubseteq$*-monotone* if, for every pair of instances D, D', such that $D \sqsubseteq D'$, we have $\mathcal{Q}(D) \sqsubseteq \mathcal{Q}(D')$. ■

Monotone queries have an interesting behavior when computing clean answers. For these queries, we can under-approximate (over-approximate) certain answers (possible answers) by taking the greatest lower bound (least upper bound) of all clean instances and then running the query on the result. Notice that we are not claiming that these are polynomial-time approximations.

Proposition 4. If $\mathcal{D}$ is a finite set of database instances and $\mathcal{Q}$ is a $\sqsubseteq$-monotone query, the following hold:

$$\mathcal{Q}(glb_{\sqsubseteq}\{D \mid D \in \mathcal{D}\}) \sqsubseteq glb_{\sqsubseteq}\{\mathcal{Q}(D) \mid D \in \mathcal{D}\}, \tag{4}$$

$$lub_{\sqsubseteq}\{\mathcal{Q}(D) \mid D \in \mathcal{D}\} \sqsubseteq \mathcal{Q}(lub_{\sqsubseteq}\{D \mid D \in \mathcal{D}\}). \tag{5}$$

Proof: For every instance $D' \in \mathcal{D}$, we clearly have $glb_{\sqsubseteq}\{D \mid D \in \mathcal{D}\} \sqsubseteq D'$, since $\mathcal{Q}$ is a monotone query, it holds $\mathcal{Q}(glb_{\sqsubseteq}\{D \mid D \in \mathcal{D}\}) \sqsubseteq \mathcal{Q}(D')$. Consequently, $Q(glb_{\sqsubseteq}\{D \mid D \in \mathcal{D}\}) \sqsubseteq glb_{\sqsubseteq}\{\mathcal{Q}(D') \mid D' \in \mathcal{D}\}$. With a similar argument, it can be shown that (5) holds. ■

Notice that we can apply Proposition 4 to the (finite) class $\mathit{Clean}(D_0, \Sigma)$ of all clean instances.

As is well known, positive relational algebra queries composed of selection, projection, Cartesian product, and union, are monotone w.r.t. $\subseteq$. However, the following example shows that monotonicity w.r.t. $\sqsubseteq$ does not hold even for very simple positive queries involving selections.

Example 7. (example 6 continued) Consider instance D_0 and the two (D_0, Σ)-clean instances D' and D''. Let $\mathcal{Q}$ be a query asking for names of people residing at “25 Main St.”, expressed in relational algebra as

$$\pi_{name}(\sigma_{address=\text{“25 Main St.”}}(R)). \tag{6}$$

Observe that $\mathcal{Q}(D_0) = \{(\text{“J. Doe”})\}$, and $\mathcal{Q}(D') = \mathcal{Q}(D'') = \emptyset$. Query $\mathcal{Q}$ is therefore not $\sqsubseteq$-monotone, because we have $D_0 \sqsubseteq D'$, $D_0 \sqsubseteq D''$, but $\mathcal{Q}(D_0) \not\sqsubseteq \mathcal{Q}(D')$, $\mathcal{Q}(D_0) \not\sqsubseteq \mathcal{Q}(D'')$. Notice that $\mathit{Cert}_{\mathcal{Q}}(D_0) = \emptyset$. ■

6.1 Lattice-sensitive operators

It is not surprising that $\sqsubseteq$-monotonicity is not satisfied by usual relational queries, in particular, by queries that *are* monotone w.r.t. set inclusion. After all, the queries we have considered so far do not even mention the $\preceq$ predicate that is at the basis of the $\sqsubseteq$ order. Next we will consider queries expressing conditions in terms of the semantic-domination lattices associated to the attribute domains, making queries sensitive to these underlying lattices. This is natural and interesting in its own right. Furthermore, we will also achieve $\sqsubseteq$-monotonicity when we replace relational selections by their natural counterparts in terms of lattice-based selection operators (cf. Section 6.2 and Example 9). In Section 6.2 these new operators will be used to relax monotone relational queries.

We introduce the (negation free) language *relaxed relational algebra*, $\mathcal{RA}_{\preceq}$, by providing two selection operators $\sigma_{a \preceq A}$ and $\sigma_{A_1 \bowtie_{\preceq} A_2}$ (for comparable attributes A_1, A_2), defined as follows.

Definition 10. The language $\mathcal{RA}_{\preceq}$ is composed of relational operators $\pi, \times, \cup$ (with usual definitions), plus $\sigma_{a \preceq A}$, and $\sigma_{A_1 \bowtie_{\preceq} A_2}$, defined on an instance D by:

(a) $\sigma_{a \preceq A}(D) = \{t^D \mid a \preceq_A t^D[A]\}$ (here $a \in \mathit{Dom}_A$),
(b) $\sigma_{A_1 \bowtie_{\preceq} A_2}(D) = \{t^D \mid \exists a \in \mathit{Dom}_A \text{ s.t. } a \preceq_A t^D[A_1], a \preceq_A t^D[A_2],\ a \neq \bot\}$. ■

For string attributes, for instance, the selection operator $\sigma_{a \preceq A}$ checks whether the value of attribute A dominates the substring a, and the join selection operator $\sigma_{A_1 \bowtie_{\preceq} A_2}$ checks whether the values of attributes A_1, A_2 dominate a common substring different from the lattice bottom element. Notice that queries in the language $\mathcal{RA}_{\preceq}$ are not domain independent: The result of posing a query to an instance depends not only on the values in the active domain of the instance but also on the domain lattices. And since those lattices emerge from matching functions, query answering depends on how data cleaning is being implemented. We claim that this is as it should be, since different implementations of data cleaning and matching functions can lead to very different answers.

It can be easily observed that all operators in the language $\mathcal{RA}_{\preceq}$ are $\sqsubseteq$-monotone: if a tuple t satisfies a selection condition, so does a tuple t' with $t \preceq t'$, and the other operators are $\sqsubseteq$-monotone for the same reason that they are $\subseteq$-monotone. Thus, every query expression in $\mathcal{RA}_{\preceq}$ that is obtained by composing these operators is also $\sqsubseteq$-monotone.

Proposition 5. Let $\mathcal{Q}$ be a query in $\mathcal{RA}_{\preceq}$. For every two instances D, D' such that $D \sqsubseteq D'$, we have $\mathcal{Q}(D) \sqsubseteq \mathcal{Q}(D')$.

Proof: We can prove the proposition by an structural induction on the relational algebra expression. It is enough to show that every operation in $\mathcal{RA}_{\preceq}$ is monotone. Projection, cartesian product, and union are clearly monotone operators w.r.t. $\sqsubseteq$. Now let D, D' be two instances such that $D \sqsubseteq D'$. Consider query $\mathcal{Q} : \sigma_{a \preceq A} R$ for relation R in the schema. Let t be an R-tuple in $\mathcal{Q}(D)$. Clearly t is an R-tuple in D. Therefore, there is an R-tuple t' in D' with $t \preceq t'$. Now it holds $a \preceq t^D[A] \preceq t'^{D'}[A]$, and thus t' is in $\mathcal{Q}(D')$.

Now consider the query $\mathcal{Q}' : \sigma_{A_1 \bowtie_{\preceq} A_2} R$, and let t be an R-tuple in $\mathcal{Q}'(D)$. Then there is $a \in \mathit{Dom}_A$ s.t. $a \preceq t^D[A_1]$, $a \preceq t^D[A_2]$, and $a \neq \bot$. Since $D \sqsubseteq D'$, there should be an R-tuple t' in D' with $t \preceq t'$. Now it holds $a \preceq t^D[A_1] \preceq t'^{D'}[A_1]$ and $a \preceq t^D[A_2] \preceq t'^{D'}[A_2]$. Therefore, t' is in $\mathcal{Q}(D')$. The inductive case where query $\mathcal{Q}$ is $\sigma_{a \preceq A}(\mathcal{Q}')$ or $\sigma_{A_1 \bowtie_{\preceq} A_2}(\mathcal{Q}')$ for a sub-query $\mathcal{Q}'$ can be similarly obtained. ■

From Propositions 4 and 5 we obtain

Theorem 4. For an instance D_0 subject to a set of MDs, and every $\sqsubseteq$-monotone query $\mathcal{Q}$, whether in $\mathcal{RA}$ or in $\mathcal{RA}_{\preceq}$, the following holds: $\mathcal{Q}(glb_{\sqsubseteq}(\mathit{Clean}(D_0, \Sigma))) \sqsubseteq \mathit{Cert}_{\mathcal{Q}}(D_0) \sqsubseteq \mathit{Poss}_{\mathcal{Q}}(D_0) \sqsubseteq \mathcal{Q}(lub_{\sqsubseteq}(\mathit{Clean}(D_0, \Sigma)))$. ■

Example 8. (example 6 continued) Consider the monotone query

$$\tilde{\mathcal{Q}} : \ \pi_{name}(\sigma_{\text{“25 Main St.”} \preceq address}(R)). \tag{7}$$

For the clean instances D', D'' it holds: $\tilde{\mathcal{Q}}(D') = \{\text{John Doe, J. Doe, Jane Doe}\}$ and $\tilde{\mathcal{Q}}(D'') = \{\text{J. Doe, Jane Doe}\}$. We obtain

$$\begin{aligned}
Cert_{\tilde{\mathcal{Q}}}(D_0) &= glb_{\preceq_{name}}\{\tilde{\mathcal{Q}}(D'), \tilde{\mathcal{Q}}(D'')\} = \tilde{\mathcal{Q}}(D') \curlywedge \tilde{\mathcal{Q}}(D'') \\
&= Red_{\preceq_{name}}(\{b' \curlywedge b'' \mid b' \in \tilde{\mathcal{Q}}(D'), b'' \in \tilde{\mathcal{Q}}(D'')\} \\
&= Red_{\preceq_{name}}(\{\text{John Doe} \curlywedge \text{J. Doe},\ \text{John Doe} \curlywedge \text{Jane Doe}, \\
&\qquad\qquad \text{J. Doe} \curlywedge \text{J. Doe},\ \text{J. Doe} \curlywedge \text{Jane Doe}, \\
&\qquad\qquad \text{Jane Doe} \curlywedge \text{J. Doe},\ \text{Jane Doe} \curlywedge \text{Jane Doe}\} \\
&= Red_{\preceq_{name}}(\{\text{J. Doe}, \text{Jane Doe}\}) = \{\text{Jane Doe}\}).
\end{aligned}$$

On the other side, in Example 4 we found that $D' \curlywedge D''$ is

$D' \curlywedge D''$	*name*	*phone*	*address*
	John Doe	(613)123 4567	Main St., Ottawa
	Jane Doe	(604)123 4567	25 Main St., Vancouver

Then, we obtain $\tilde{\mathcal{Q}}(glb_{\sqsubseteq}\{D', D''\}) = \tilde{\mathcal{Q}}(D' \curlywedge D'') = \{\text{Jane Doe}\}$, which coincides with $Cert_{\tilde{\mathcal{Q}}}(D_0)$. ■

In the proof of Theorem 3 we use a monotone relational query. As a consequence, we obtain the following theorem.

Theorem 5. Certain query answering for $\sqsubseteq$-monotone queries is *coNP*-complete (in the size of the initial instance D_0). ■

Corollary 1. Certain query answering for queries in $\mathcal{RA}_{\preceq}$ is *coNP*-complete (in the size of the initial instance D_0). ■

6.2 Query relaxation

As shown in Example 7, we may not get natural and expected clean answers by running a usual relational algebra query on an instance subject to matching dependencies. In particular, the usual relational selection operator uses conditions that are too strong to satisfy. As we saw in Section 6.1 it is not sensitive to the underlying lattice-theoretic structures on the domain.

We therefore propose to *relax* the queries, by taking advantage of the underlying $\preceq_A$-lattice structures obtained from matching functions, to make them $\sqsubseteq$-monotone. In this way, we achieve two goals: First, the resulting queries provide more informative answers; and second, we can approximate clean answers from above (cf. Corollary 2 below).

Now suppose that we have a query $\mathcal{Q}$, written in positive relational algebra, i.e., composed of $\pi, \times, \cup, \sigma_{A=a}, \sigma_{A_1=A_2}$, the last two being *hard* selection conditions, which is to be posed to an instance D_0. After cleaning D_0 by enforcing a set of MDs Σ to obtain a (D_0, Σ)-clean instance D, running query $\mathcal{Q}$ on D may no longer provide the expected answer, because some of the values have changed in D, i.e., they have semantically grown w.r.t. $\preceq$.

In order to capture this semantic growth, our query relaxation framework proposes the following:

Query rewriting methodology: Given a query $\mathcal{Q}$ in positive RA, transform it into a query $\mathcal{Q}_{\preceq}$ in $\mathcal{RA}_{\preceq}$, the *relaxed rewriting* of $\mathcal{Q}$, by:

(a) replacing operator $\sigma_{A=a}$ by $\sigma_{a \preceq A}$; and
(b) replacing operator $\sigma_{A_1=A_2}$ by $\sigma_{A_1 \bowtie_{\preceq} A_2}$.

The following result follows from the construction of the relaxed query.

Proposition 6. For every positive relational algebra query $\mathcal{Q}$ and instance D, we have $\mathcal{Q}(D) \sqsubseteq \mathcal{Q}_{\preceq}(D)$, where $\mathcal{Q}_{\preceq}$ is the relaxed rewriting of $\mathcal{Q}$. ■

Corollary 2. For an instance D_0 subject to a set of MDs, and every positive relational algebra query $\mathcal{Q}$, we have $\mathit{Cert}_{\mathcal{Q}}(D_0) \sqsubseteq \mathit{Cert}_{\mathcal{Q}_{\preceq}}(D_0)$, and $\mathit{Poss}_{\mathcal{Q}}(D_0) \sqsubseteq \mathit{Poss}_{\mathcal{Q}_{\preceq}}(D_0)$.

Proof: From Proposition 6, $\{\mathcal{Q}(D) \mid D \in \mathit{Clean}(D_0, \Sigma)\} \sqsubseteq \{\mathcal{Q}_{\preceq}(D) \mid D \in \mathit{Clean}(D_0, \Sigma)$. Now, taking $\mathit{glb}_{\sqsubseteq}$ on both sides, and next also $\mathit{lub}_{\sqsubseteq}$ on both sides, we obtain the two conclusions, respectively. ■

Example 9. (example 8 continued) Consider again instances D_0 and the (D_0, Σ)-clean instances D' and D'', and query $\mathcal{Q}$ asking for names of people residing at "25 Main St.", expressed as $\pi_{name}(\sigma_{address=\text{"25 Main St."}}(R))$. This is query (6) in Example 7, where we obtained the empty answer from each of D', D''. So, in this case we have $\mathit{Cert}_{\mathcal{Q}}(D_0) = \mathit{Poss}_{\mathcal{Q}}(D_0) = \emptyset$, not a very informative outcome.

However, after the relaxation of $\mathcal{Q}$, we obtain the monotone query $\mathcal{Q}_{\preceq}$: $\pi_{name}(\sigma_{\text{"25 Main St."} \preceq address}(R))$, which is query $\tilde{\mathcal{Q}}$ in (7) in Example 8, where we obtained

$$\mathcal{Q}_{\preceq}(D') = \{\text{John Doe}, \text{J. Doe}, \text{Jane Doe}\},$$
$$\mathcal{Q}_{\preceq}(D'') = \{\text{J. Doe}, \text{Jane Doe}\},$$

and also $\mathit{Cert}_{\mathcal{Q}_{\preceq}}(D_0) = \{\text{Jane Doe}\}$. This outcome is much more informative than the one obtained from $\mathcal{Q}$; and, above all, is sensitive to the underlying information lattice. ■

7 Approximating Clean Answers

Given the high computational cost of clean query answering when there are multiple clean instances, it would be desirable to provide an approximation to clean answers that is computable in polynomial time. In this section, we are interested in approximating clean answers by producing an under-approximation of certain answers and an over-approximation of possible answers for a given $\sqsubseteq$-monotone query $\mathcal{Q}$. Remember that, by Theorem 5, we know that clean query answering for monotone queries is *coNP*-complete. As a consequence, approximating clean query answering is a natural and relevant problem. That is, we would like to obtain sets of query answers (instances) $\mathcal{Q}_{\downarrow}(D_0), \mathcal{Q}_{\uparrow}(D_0)$, such that $\mathcal{Q}_{\downarrow}(D_0) \sqsubseteq \mathit{Cert}_{\mathcal{Q}}(D_0)$ and $\mathit{Poss}_{\mathcal{Q}}(D_0) \sqsubseteq \mathcal{Q}_{\uparrow}(D_0)$.

Since $\mathcal{Q}$ is a monotone query, by Proposition 4, we have

$$\mathcal{Q}(\mathit{glb}_{\sqsubseteq}\{D \mid D \text{ is } (D_0, \Sigma)\text{-clean}\}) \sqsubseteq \mathit{Cert}_{\mathcal{Q}}(D_0), \tag{8}$$

and moreover,

$$\mathit{Poss}_{\mathcal{Q}}(D_0) \sqsubseteq \mathcal{Q}(\mathit{lub}_{\sqsubseteq}\{D \mid D \text{ is}(D_0, \Sigma)\text{-clean}\}). \tag{9}$$

In consequence, it is good enough to find an under-approximation $D_{\downarrow}$ for the greatest lower bound in (8) and an over-approximation $D_{\uparrow}$ for the least upper bound in (9); and then pose $\mathcal{Q}$ to these approximations to obtain $\mathcal{Q}_{\downarrow}(D_0)$ and $\mathcal{Q}_{\uparrow}(D_0)$.

The reason for having multiple clean instances is that matching dependencies are not necessarily interaction-free and the matching functions are not necessarily similarity preserving. Intuitively speaking, we can under-approximate the greatest lower bound of clean instances by not enforcing some of the interacting MDs. On the other side, we can over-approximate the least upper bound by assuming that the matching functions are similarity preserving. This would lead us to keep applying MDs on the assumption that unresolved similarities still persist. We present two chase-like procedures to compute two instances $D_{\downarrow}$ and $D_{\uparrow}$ corresponding to these approximations.

7.1 Under-approximating the greatest lower bound

To provide an under-approximation for the greatest lower bound of all clean instances, we provide a new chase-like procedure, which enforces only MDs that are enforced in every clean instance. These MDs are applicable to those initial similarities that exist in the original dirty instance, which are never broken by enforcing other MDs during any chase procedure of producing a clean instance.

Let Σ be a set of MDs, and $\varphi, \varphi' \in \Sigma$. We say that φ *precedes* φ' if the set of attributes on the left-hand side of φ' contains the attribute on the right-hand side of φ. We say that φ *interacts with* φ' if there are MDs $\varphi_1, \ldots, \varphi_k \in \Sigma$, such that φ precedes φ_1, φ_k precedes φ', and φ_i precedes φ_{i+1} for $i \in [1, k-1]$, i.e., the interaction relationship can be seen as the transitive closure of precedence relationship.

Let D_0 be a dirty database instance. Let $\varphi : R_1[X_1] \approx R_2[X_2] \rightarrow R_1[A_1] \rightleftharpoons R_2[A_2]$ be an MD in Σ. We say φ is *freshly applicable* on t_1, t_2 in D_0 if $t_1^{D_0}[X_1] \approx t_2^{D_0}[X_2]$, and $t_1^{D_0}[A_1] \neq t_2^{D_0}[A_2]$. We say φ is *safely applicable* on t_1, t_2 in D_0 if φ is freshly applicable on t_1, t_2 in D_0, and for every $\varphi' \in \Sigma$ that interacts with φ, φ' is not freshly applicable on t_1, t_3 or t_2, t_3 in D_0 for any tuple t_3 (see Example 10).

Definition 11. For an instance D_0 and a set of MDs Σ, an instance D_k is (D_0, Σ)-*under clean* if there exists a finite sequence of instances $D_1, \ldots, D_{k-1}$, such that

1. For every $i \in [1,k]$, $(D_{i-1}, D_i)_{[t_1^i, t_2^i]} \models \varphi^i$, for some $\varphi^i \in \Sigma$ and tuple identifiers t_1^i, t_2^i, such that φ^i is safely applicable on t_1^i, t_2^i in D_0.
2. For every MD $\varphi : R_1[X_1] \approx R_2[X_2] \rightarrow R_1[A_1] \rightleftharpoons R_2[A_2]$ in Σ and tuples t_1, t_2, such that φ is safely applicable on t_1, t_2 in D_0, we have $t_1^{D_k}[A_1] = t_2^{D_k}[A_2]$. ■

Definition 11 characterizes a chase-based procedure that keeps enforcing MDs that are safely applicable in the original dirty instance until all such MDs are enforced. Notice that an under clean instance may not be stable. Moreover, safely applicable MDs never interfere with each other, in the sense that enforcing one of them never breaks the initial similarities in the dirty instance that are needed for enforcing other safely applicable MDs.

Proposition 7. For every instance D_0 and every set of MDs Σ, there is a unique (D_0, Σ)-under clean instance $D_{\downarrow}$. ■

The proof of this proposition is very similar to that of Proposition 3. It immediately follows from the following lemma.

Lemma 4. Let $D_1, \ldots, D_k$ be a sequence of instances for deriving an (D_0, Σ)-under clean instance $D_{\downarrow}$ (as in Definition 11). Let D be any (D_0, Σ)-under clean instance, not necessarily equal to $D_{\downarrow}$. Then, for every $i \in [0,k]$, it holds

1. $t_1^{D_i}[X_1] = t_1^{D_0}[X_1]$ and $t_2^{D_i}[X_2] = t_2^{D_0}[X_2]$, for every tuple identifiers t_1, t_2, where X_1, X_2 are the lists of attributes on the left-hand side of φ^i.
2. $t^{D_i}[A] \preceq t^{D}[A]$, for every tuple identifier t and every attribute A.

Proof: For 1., suppose that for some $i \in [0,k]$, $t_1^{D_i}[X_1] \neq t_1^{D_0}[X_1]$. Then there exists $j < i$, tuple t_3, and MD $\varphi^j \in \Sigma$, such that $(D_{j-1}, D_j)_{[t_1, t_3]} \models \varphi^j$, with attribute $B_1 \in X_1$ on the right-hand side of φ^j. MD φ^j has to be safely applicable on t_1, t_3 in D_0, which means that φ^i cannot be safely applicable on t_1, t_2 in D_0, a contradiction. The proof of 2. is similar to the proof of 2. in Lemma 3. ■

Clearly, an under clean instance $D_{\downarrow}$ can be computed in polynomial time in the size of the dirty instance D_0. To construct it, we first need to identify safely applicable MDs in D_0, and then enforce them in any arbitrary order until no such MDs can be enforced. Next we show that $D_{\downarrow}$ is an under-approximation to every (D_0, Σ)-clean instance. Intuitively, this is because $D_{\downarrow}$ is obtained by enforcing MDs that are enforced in every chase-based procedure of producing a clean instance.

Proposition 8. (*soundness of under-approximation*) For every instance D_0 and every set of MDs Σ, for the (D_0, Σ)-under clean instance $D_{\downarrow}$ and every (D_0, Σ)-clean instance D, it holds $D_{\downarrow} \sqsubseteq D$. ■

The proof of this proposition follows from the following two lemmas.

Lemma 5. Let D_0 be an instance subject to a set of MDs Σ, and D be a (D_0, Σ)-clean instance. For every two tuples t_1, t_2 and MD $\varphi : R_1[X_1] \approx R_2[X_2] \rightarrow R_1[A_1] \rightleftharpoons R_2[A_2]$ in Σ, such that φ is safely applicable on t_1, t_2 in D_0, it holds $t_1^D[X_1] = t_1^{D_0}[X_1]$ and $t_2^D[X_2] = t_2^{D_0}[X_2]$. ■

Lemma 6. Let D_0 be an instance subject to a set of MDs Σ, D be a (D_0, Σ)-clean instance, $D_\downarrow$ be the (D_0, Σ)-under clean instance; and $D_1, \ldots, D_k$ be a sequence of instances for deriving $D_\downarrow$ (as in Definition 11). Then, for every $i \in [0, k]$, it holds $t^{D_i}[A] \preceq t^D[A]$, for every tuple identifier t and every attribute A. ■

The proof of this lemma is by induction on i; and, not surprisingly, is very similar to the proof of 2. in Lemma 3, which applies to interaction-free sets of MDs. From Proposition 8 we immediately obtain

Corollary 3. If $D_\downarrow$ is a (D_0, Σ)-under clean instance, then
$$D_\downarrow \sqsubseteq glb_{\sqsubseteq}(Clean(D_0, \Sigma)).$$ ■

Notice that an arbitrary (D_0, Σ)-clean instance D may not be a sound under-approximation for every other (D_0, Σ)-clean instances D', because $D \sqsubseteq D'$ may not hold.

From Theorem 4 and Corollary 3 we immediately obtain the following result.

Theorem 6. If $D_\downarrow$ is a (D_0, Σ)-under clean instance, then for every monotone query $\mathcal{Q}$, it holds $\mathcal{Q}(D_0) \sqsubseteq \mathcal{Q}(D_\downarrow) \sqsubseteq \mathcal{Q}(glb_{\sqsubseteq}(Clean(D_0, \Sigma))) \sqsubseteq Cert_{\mathcal{Q}}(D_0)$. ■

Example 10. (Example 5 continued) For the given instance D_0 and set of MDs Σ, observe that MD φ_1 is safely applicable on the first and second tuples in D_0. Moreover, φ_2 is freshly applicable, but not safely applicable on the second and third tuples. Accordingly, we obtain (D_0, Σ)-under clean instance $D_\downarrow$, shown below, by enforcing φ_1 on the first two tuples.

$D_\downarrow$	A	B	C
	a_1	$\langle b_1, b_2 \rangle$	c_1
	a_2	$\langle b_1, b_2 \rangle$	c_2
	a_3	b_3	c_3

Notice that for the two (D_0, Σ)-clean instances D_2, D_3' in Example 5, we have $D_\downarrow \sqsubseteq D_2$ and $D_\downarrow \sqsubseteq D_3'$. Also notice that $D_\downarrow$ is not a stable instance. Now consider the query $\mathcal{Q} : \pi_C(\sigma_{A=a_2} R)$. This query behaves monotonically for our purpose, because the values of attribute A are not changing by enforcing MDs. If we pose $\mathcal{Q}$ to $D_\downarrow$, we obtain $\mathcal{Q}(D_\downarrow) = \{c_2\}$. Observe that $Cert_{\mathcal{Q}}(D_0) = \{\langle c_1, c_2 \rangle\}$, and thus $\mathcal{Q}(D_\downarrow)$ provides an under-approximation for $Cert_{\mathcal{Q}}(D_0)$. This example also shows that an arbitrary clean instance, D_3' here, may not provide a sound approximation to certain answer since $\mathcal{Q}(D_3') = \{\langle c_1, c_2, c_3 \rangle\} \not\sqsubseteq Cert_{\mathcal{Q}}(D_0)$. ■

7.2 Over-approximating the least upper bound

To provide an over-approximation for the least upper bound of all clean instances, we *modify* every similarity relation so that the corresponding matching function becomes similarity preserving. For a similarity relation $\approx_A$ and the corresponding matching function m_A, we define $\approx_A^*$ as follows: For every $a, a' \in Dom_A$, $a \approx_A^* a'$ iff there is $a'' \in Dom_A$, such that $a \approx_A a''$ and $m_A(a', a'') = a'$. Given a set of MDs Σ, we obtain Σ^* by replacing every similarity relation $\approx_A$ in the MDs by $\approx_A^*$.

Notice that the relation $\approx_A^*$ is well defined in the sense that a and a' are interchangeable. Secondly, it should be obvious that the matching function m_A is similarity preserving w.r.t. the relation $\approx_A^*$.

Definition 12. For an instance D_0 and a set of MDs Σ, an instance $D_\uparrow$ is (D_0, Σ)-*over clean* if it is (D_0, Σ^*)-clean. ■

Proposition 9. For every instance D_0 and every set of MDs Σ, there is a unique (D_0, Σ)-over clean instance $D_\uparrow$. Moreover, $D_\uparrow$ can be computed in polynomial time in the size of D_0.

Proof: The first claim follows from Proposition 2, because we are transforming a set Σ of MDs into a set Σ^* that uses similarity preserving matching functions. For the second claim, to construct $D_\uparrow$, we first need to obtain Σ^*, as described above, and enforce MDs in Σ^* in any arbitrary order until getting a stable instance w.r.t. Σ^*. ■

Next we show that $D_\uparrow$ is an over-approximation for every (D_0, Σ)-clean instance. Intuitively, this is because $D_\uparrow$ is obtained by enforcing (at least) all MDs that are present in any chase-like procedure of producing a clean instance.

Proposition 10. (*completeness of over-approximation*) Let D_0 be an instance subject to a set of MDs. For the (D_0, Σ)-over clean instance $D_\uparrow$ and every (D_0, Σ)-clean instance D, it holds $D \sqsubseteq D_\uparrow$. ■

Notice again that an arbitrary (D_0, Σ)-clean instance D may not be an over-approximation for every other (D_0, Σ)-clean instance D', because $D' \sqsubseteq D$ may not hold.

From Propositions 10 and 4, we immediately obtain the following result.

Theorem 7. Let D_0 be an instance subject to a set of MDs, and $D_\uparrow$ be the (D_0, Σ)-over clean instance. Then, for every monotone query $\mathcal{Q}$, it holds

$$Poss_{\mathcal{Q}}(D_0) \sqsubseteq \mathcal{Q}(lub_{\sqsubseteq}(Clean(D_0, \Sigma))) \sqsubseteq \mathcal{Q}(D_\uparrow).$$ ■

Example 11. (Example 10 continued) By assuming that old similarities hold after applying matching functions (e.g., $\langle b_1, b_2 \rangle \approx^* b_3$), we obtain the (D_0, Σ)-over clean instance $D_\uparrow$ shown below.

$D_\uparrow$	A	B	C
	a_1	$\langle b_1, b_2\rangle$	$\langle c_1, c_2, c_3\rangle$
	a_2	$\langle b_1, b_2\rangle$	$\langle c_1, c_2, c_3\rangle$
	a_3	b_3	$\langle c_1, c_2, c_3\rangle$

Notice that for the two (D_0, Σ)-clean instances D_2, D_3' in Example 5, we have $D_2 \sqsubseteq D_\uparrow$ and $D_3' \sqsubseteq D_\uparrow$. If we pose query $\mathcal{Q} : \pi_C(\sigma_{A=a_2}R)$ to $D_\uparrow$, we obtain $\mathcal{Q}(D_\uparrow) = \{\langle c_1, c_2, c_3\rangle\}$. Observe that $Poss_{\mathcal{Q}}(D_0) = \{\langle c_1, c_2, c_3\rangle\}$, and thus $\mathcal{Q}(D_\uparrow)$ provides an over-approximation for $Poss_{\mathcal{Q}}(D_0)$. It can be seen that an arbitrary (D_0, Σ)-clean instance, say D_2 for instance, may not provide a complete approximation to possible answer since $Poss_{\mathcal{Q}}(D_0) \not\sqsubseteq \mathcal{Q}(D_2) = \{\langle c_1, c_2\rangle\}$. ■

8 The Swoosh's Entity Resolution Connection

In [9], a generic conceptual framework for entity resolution is introduced. It considers a general match relation M, which is close to our similarity predicates $\approx$, and a general merge function, μ, which is close to our $\mathfrak{m}$ functions. In this section we establish a connection between our MD framework and Swoosh.

A full comparison between our framework and Swoosh has its subtleties due to the differences between these frameworks, for example: (a) Swoosh works at the *record level*, and MDs at the attribute level. (b) Swoosh does not use tuple identifiers and some tuples may be discarded, those that are dominated by others in the instance. The main problem is (a).

We make a comparison, or better, we reconstruct Swoosh in the MD framework, by considering first, in Section 8.1, a general, attribute-free version of Swoosh, and next, in Section 8.2, a particular – but still general enough – case of Swoosh, namely the combination of the *union case* with *merge domination* that does consider attributes. These embeddings of Swoosh into our MD framework give additional evidence for the strength of the latter.

8.1 MDs and general Swoosh

Here we follow Swoosh's general abstraction, where the match relation M and the merge function μ are defined at the *record* level. That is, when two records in a database instance are matched (found similar), we can merge them into a new record. We keep doing this until the entity resolution of the instance is computed. In this section we establish a connection between our MD framework and Swoosh framework.

Swoosh views a database instance I as a finite set of records $I = \{r_1, \ldots, r_n\}$ taken from an infinite domain of records Rec. Relation M maps $Rec \times Rec$ into $\{true, false\}$. When two records are similar and have to be merged, M takes the value $true$. Moreover, μ is a partial function from $Rec \times Rec$ into Rec. It produces the merge of two records into a new record, and is defined only when M takes the value $true$.

Given an instance I, the *merge closure* of I is defined as the smallest set of records $\bar{I}$, such that $I \subseteq \bar{I}$, and, for every two records r_1, r_2 for which $M(r_1, r_2) =$ *true*, we have $\mu(r_1, r_2) \in \bar{I}$. The merge closure of an instance is unique and can be obtained by adding merges of matching records until a fixpoint is reached.

Swoosh considers a general domination relationship between two records r_1, r_2, written as $r_1 \preceq_s r_2$, which means the information in r_1 is subsumed by the information in r_2. Then for two instances I_1, I_2, we write $I_1 \sqsubseteq_s I_2$ whenever every record of I_1 is dominated by some record in I_2. Notice, we use the subscript s for $\preceq_s$ and $\sqsubseteq_s$ in Swoosh to avoid confusion with the $\preceq$ and $\sqsubseteq$ symbols introduced and used in the previous sections.

For an instance I, an *entity resolution* is defined as a subset-minimal set of records I', such that $I' \subseteq \bar{I}$ and $\bar{I} \sqsubseteq_s I'$. It is shown that for every instance I, there is a unique entity resolution I' [9], which can be obtained from the merge closure by removing records that are dominated by other records.

Here we are interested in the Swoosh case where match relation M is reflexive and symmetric, and the merge function μ is idempotent, commutative, and associative. We then use the domination order imposed by the merge function, which is defined by: $r_1 \preceq_s r_2$ if and only if $\mu(r_1, r_2) = r_2$. Under these assumptions, the merge closure and therefore the entity resolution of every instance are finite [9].[5]

Now we reconstruct the Swoosh framework using matching dependencies. We assume that records in a Swoosh instance I are taken from a relation $R(A)$ with the *single* attribute A. This is to make sure that comparing and merging records are done at the record level. Attribute A in the relation $R(A)$ could be thought of as a complex-type attribute containing multiple atomic attributes of a record (cf. Section 8.2). Notice that this is an abstraction and not a restriction. That is, we can still evaluate the similarity of two records based on the similarity of individual atomic attribute values, and merge two records by merging pairwise atomic attribute values.

Given a Swoosh instance $I = \{r_1, \ldots, r_n\}$, we introduce tuple identifiers, and construct a relational instance $D_0 = \{t_i \mid t_i$ is a unique tuple identifier and $t_i[A] = r_i\}$. Furthermore, we let the set of matching dependencies Σ contain only one MD:

$$\varphi : R[A] \approx R[A] \rightarrow R[A] \rightleftharpoons R[A].$$

We let the similarity relation $\approx$ be equal to Swoosh's match relation M, and the matching function m_A to be equal to Swoosh's merge function μ. Clearly, our partial orders $\preceq$ and $\sqsubseteq$ used in the previous sections now precisely coincide with Swoosh's partial orders $\preceq_s$ and $\sqsubseteq_s$. We therefore drop the subscript s hereafter.

Example 12. Consider a Swoosh instance $I = \{r_1, r_2, r_3\}$, where two similarities hold: $M(r_1, r_2) =$ *true* and $M(r_2, r_3) =$ *true*. Let $\langle r_1, r_2\rangle$ and $\langle r_2, r_3\rangle$ denote $\mu(r_1, r_2)$ and $\mu(r_2, r_3)$, resp. Also assume that $M(\langle r_1, r_2\rangle, \langle r_2, r_3\rangle) =$ *true*; and let

[5] Finiteness is shown for the case when match and merge have the *representativity* property (equivalent to being similarity preserving) in addition to other properties. However, the proof in [9] can be modified so that representativity is not necessary.

$\langle r_1, r_2, r_3\rangle$ denote $\mu(\langle r_1, r_2\rangle, \langle r_2, r_3\rangle)$, the result of merging $\langle r_1, r_2\rangle$ and $\langle r_2, r_3\rangle$. The figure below shows instance I, its merge closure $\bar{I}$, and its entity resolution I'.

I
r_1
r_2
r_3

$\bar{I}$
r_1
r_2
r_3
$\langle r_1, r_2\rangle$
$\langle r_2, r_3\rangle$
$\langle r_1, r_2, r_3\rangle$

I'
$\langle r_1, r_2, r_3\rangle$

To illustrate the computations involved, notice that, e.g., $\mu(r_1, \langle r_1, r_2\rangle) = \mu(\langle r_1, r_1\rangle, r_2) = \mu(r_1, r_2) = \langle r_1, r_2\rangle$. Then, $r_1 \preceq \langle r_1, r_2\rangle$. Similarly, $\mu(\langle r_1, r_2\rangle, \langle r_1, r_2, r_3\rangle) = \langle r_1, r_2, r_3\rangle$, and then, $\langle r_1, r_2\rangle \preceq \langle r_1, r_2, r_3\rangle$.

As described above, from I, we construct an instance D_0 of $R(A)$, and we let Σ contain the single MD $\varphi : R[A] \approx R[A] \rightarrow R[A] \rightleftharpoons R[A]$. The following similarities hold $r_1 \approx r_2$, $r_2 \approx r_3$, and $\langle r_1, r_2\rangle \approx \langle r_2, r_3\rangle$. We also have $m_A(r_1, r_2) = \langle r_1, r_2\rangle$, $m_A(r_2, r_3) = \langle r_2, r_3\rangle$, and $m_A(\langle r_1, r_2\rangle, \langle r_2, r_3\rangle) = \langle r_1, r_2, r_3\rangle$. With these elements we obtain two (D_0, Σ)-clean instances D' and D'':[6]

D_0	A
t_1	r_1
t_2	r_2
t_3	r_3

D'	A
t_1	$\langle r_1, r_2\rangle$
t_2	$\langle r_1, r_2\rangle$
t_3	r_3

D''	A
t_1	r_1
t_2	$\langle r_2, r_3\rangle$
t_3	$\langle r_2, r_3\rangle$

■

We can naturally compare a Swoosh instance with an instance with tuple identifiers w.r.t. partial order $\sqsubseteq$. In the example above, $D' \sqsubseteq I'$ holds, because for every tuple $t^{D'}$ (in D'), there is a record r in I' such that $t[A]^{D'} \preceq r$. This suggests a relationship between the unique Swoosh entity resolution I' of instance I and an arbitrary (D_0, Σ)-clean instance D.

Theorem 8. Let D_0 and Σ be associated to record instance I. For every (D_0, Σ)-clean instance D, and the Swoosh entity resolution I', it holds $D \sqsubseteq I'$.

For the proof of Theorem 8, we need the following lemma.

Lemma 7. Let D_0 and $\Sigma = \{\varphi\}$ be associated to record instance I, and let $D_1, \ldots, D_k$ be a sequence of instances such that, for every $i \in [1, k]$, $(D_{i-1}, D_i)_{[t_1^i, t_2^i]} \models \varphi$ for two tuple identifiers t_1^i, t_2^i. Let $\bar{I}$ be the merge closure of I. Then for every $i \in [0, k]$, $D_i \subseteq \bar{I}$ holds. More precisely, for every tuple t in D_i, there is a record r in $\bar{I}$ such that $t[A] = r$.

Proof: The proof of this lemma is by an induction on i. For $i = 0$, we have $I \subseteq \bar{I}$ by definition of merge closure, and thus $D_0 \subseteq \bar{I}$ clearly holds. Suppose that for $j < i$, $D_j \subseteq \bar{I}$ holds. Now consider the instance D_i and let

[6] Notice that the single MD does not form an interaction-free set of MDs.

t_1, t_2 be the only two tuples that have changed during the transition from D_{i-1} to D_i. That is $(D_{i-1}, D_i)_{[t_1,t_2]} \models \varphi$. We then have $t_1^{D_{i-1}}[A] \approx t_2^{D_{i-1}}[A]$, and $t_1^{D_i}[A] = t_2^{D_i}[A] = m_A(t_1^{D_{i-1}}[A], t_2^{D_{i-1}}[A])$. Moreover, by the induction hypothesis, $t_1^{D_{i-1}}[A], t_2^{D_{i-1}}[A]$ are equal to two records r, r' in $\bar{I}$, respectively, and $M(r, r') = \mathit{true}$ (the two records are similar and matched). By definition of merge closure, $\bar{I}$ should contain a record $\langle r, r' \rangle$ corresponding to the result of merging r, r'. Notice that $t_1^{D_i}[A] = t_2^{D_i}[A] = \langle r, r' \rangle$ since the result of applying the matching function m_A to $t_1^{D_{i-1}}[A], t_2^{D_{i-1}}[A]$ is the same as the result of applying the merge function μ to r, r'. Thus, $D_i \subseteq \bar{I}$. ■

Proof of Theorem 8: Since D is a clean instance, there is a chase sequence for it. From Lemma 7, we obtain $D \subseteq \bar{I}$, where $\bar{I}$ is the merge closure of instance I. By definition, for the merge closure $\bar{I}$ and the entity resolution I' it holds $\bar{I} \sqsubseteq I'$. Thus, $D \sqsubseteq I'$ holds. ■

From Theorem 8, we immediately conclude that the Swoosh entity resolution I' dominates the least upper bound of all clean instances. That is, $\mathit{lub}_{\sqsubseteq}(\mathit{Clean}(D_0, \varSigma)) \sqsubseteq I'$. An interesting question is whether the reverse is also true, i.e., whether the entity resolution is actually equivalent to the least upper bound of all clean instances. The following example shows that this does not hold.

Example 13. (example 12 continued) Consider the instances D_0, D' and D''. The following instance shows the result of computing the least upper bound of D' and D'', which is obtained by taking the union of the two instances and removing tuples that are dominated by other tuples.

$D' \curlyvee D''$	A
	$\langle r_1, r_2 \rangle$
	$\langle r_2, r_3 \rangle$

Comparing this instance with Swoosh entity resolution I' in Example 12, we can easily observe that $D' \curlyvee D'' \sqsubseteq I'$, but $I' \not\sqsubseteq D' \curlyvee D''$ (assuming that $\langle r_1, r_2, r_3 \rangle$ is different from $\langle r_1, r_2 \rangle$ and $\langle r_2, r_3 \rangle$). ■

Corollary 4. For $D_0, \varSigma$ associated to a record instance I, the Swoosh entity resolution I' is an over-approximation for the least upper bound of all clean instances, i.e., $\mathit{lub}_{\sqsubseteq}(\mathit{Clean}(D_0, \varSigma)) \sqsubseteq I'$. However, the reverse does not necessarily hold. ■

8.2 MDs and the union case for Swoosh

In this section we assume that records as conceived by Swoosh correspond to ground tuples of a single relational predicate, say R. In consequence, Rec denotes the set of ground tuples of the form $R(\bar{s})$. If the attributes of R are $A_1, \ldots, A_n$, then the component s_i of $\bar{s}$ belongs to an underlying domain Dom_{A_i}.

As in the previous section, relation M maps $Rec \times Rec$ into $\{true, false\}$; and μ is a partial function from $Rec \times Rec$ into Rec. It is defined only when M takes the value *true*.

Now, the *union case* for Swoosh [9, sec. 2] arises when the merge function μ produces the union of the records, defined as the component-wise union of attribute values. This latter union makes sense if the attribute values are sets of values from an even deeper data domain.

This case can be seen as a special case of the general case described in Section 8.1, by considering the generic auxiliary attribute A there as a complex attribute that represents the attributes $A_1, \ldots, A_n$ we are considering here. Due to the intrinsic interest in, and subtleties and technical details of the union case, we are presenting here a direct MD-based reconstruction of Swoosh for this case.

For each of the n attributes A_i of R, we consider a possibly denumerable domain D_{A_i} (repeated attributes in R share the same domain, but it is conceptually simpler to assume that attributes are all different). Each D_{A_i} has a similarity relation $\approx_{A_i}$, which is reflexive and symmetric. Now, for each attribute A_i of R, its domain becomes $Dom_{A_i} := \cup_{k \in \mathbb{N}} \mathcal{P}^k(D_{A_i})$, where $k > 0$ and $\mathcal{P}^k(D_{A_i})$ denotes the set of subsets of D_{A_i} of cardinality k. Thus, the elements of Rec are of the form $R(s_1, \ldots, s_n)$, with each s_i being a set that belongs to Dom_{A_i}. An initial instance D, before any entity resolution, will be a finite subset of Rec, and each attribute value in a record, say s_i for A_i, will be a singleton of the form $\{a_i\}$, with $a_i \in D_{A_i}$.

The $\approx_{A_i}$ relation on D_{A_i} induces a similarity relation $\approx_{\{A_i\}}$ on Dom_{A_i}, as follows: $s_1 \approx_{\{A_i\}} s_2$ holds iff there exist $a_1 \in s_1, a_2 \in s_2$ with $a_1 \approx_{A_i} a_2$. Each $\approx_{\{A_i\}}$ is reflexive and symmetric. ($s \approx_{\{A_i\}} s$, because there is $a \in s$ and $\approx_{A_i}$ is reflexive; and symmetry follows from the symmetry of $\approx_{A_i}$.) We also consider matching functions $m_{\{A_i\}} : Dom_{A_i} \times Dom_{A_i} \to Dom_{A_i}$ defined by $m_{\{A_i\}}(s_1, s_2) := s_1 \cup s_2$. The structures $\langle Dom_{A_i}, \approx_{\{A_i\}}, m_{\{A_i\}} \rangle$ have all the properties described in Sections 2 and 3.

Proposition 11. Each matching function $m_{\{A_i\}}$ is total, idempotent, commutative and associative. It is also similarity preserving w.r.t. the $\approx_{\{A_i\}}$ similarity relation.

Proof: In fact: If $s_1 \approx_{\{A\}} s_2$, then there are $a_1 \in s_1, a_2 \in s_2$ with $a_1 \approx_A a_2$. Since a_2 also belongs to $s_2 \cup s_3$, for every $s_3 \in Dom_A$, it holds $s_2 \cup s_3 = m_{\{A\}}(s_2, s_3) \approx_{\{A\}} s_1$. ■

Now, based on [9] (cf. proof of proposition 2.4 in it), we are ready to define the "union match and merge case" for Swoosh. Consider two elements of Rec, say $r_1 = R(\bar{s}^1), r_2 = R(\bar{s}^2)$: (a) $M(r_1, r_2) := true$ iff for some i, $s_i^1 \approx_{\{A_i\}} s_i^2$. (b) When $M(r_1, r_2) := true$, $\mu(r_1, r_2) := R(m_{\{A_1\}}(s_1^1, s_1^2), \ldots,\ m_{\{A_n\}}(s_n^1, s_n^2))$.

Function M is reflexive and commutative, which follows from the reflexivity and symmetry of the $\approx_{\{A\}}$. From [9, Prop. 2.4] we obtain that the combination of M and μ has Swoosh's ICAR properties, namely:[7]

I^s: Idempotency: $\forall r \in \mathit{Rec}, M(r,r)$ holds, and $\mu(r,r) = r$.
C^s: Commutativity: $\forall r_1, r_2 \in \mathit{Rec}, M(r_1,r_2)$ iff $M(r_2,r_1)$. Also $M(r_1,r_2)$ implies $\mu(r_1,r_2) = \mu(r_2,r_1)$.
A^s: Associativity: $\forall r_1, r_2, r_3 \in \mathit{Rec}$, if $\mu(r_1,\mu(r_2,r_3))$ and $\mu(\mu(r_1,r_2),r_3)$ exist, then they are equal.
R^s: Representativity: $\forall r_1, r_2, r_3, r_4 \in \mathit{Rec}$, if $r_3 = \mu(r_1,r_2)$ and $M(r_1,r_4)$ holds, then $M(r_3,r_4)$ also holds.

Now, Swoosh framework with M and μ on Dom_A can be reconstructed by means of the following set Σ^S of MDs: For $1 \leq i,j \leq n$,

$$R[A_i] \ \approx_{\{A_i\}} R[A_i] \ \longrightarrow \ R[A_j] \rightleftharpoons R[A_j]. \tag{10}$$

The RHS of (10) has to be applied, as expected, with the matching functions $m_{\{A_j\}}$. From Propositions 2 and 11, we obtain that there is a single (D, Σ^S)-clean instance D^m. Consistently with our MD framework, we will assume that records have tuple identifiers. Actually, in order to make the comparison between the two frameworks clearer, in this section and for the MD framework, we will use explicit tuple ids. They will be positioned in the first, extra attribute of each relation. When the MDs are applied, only the new version of a tuple is kept.

In the case of Swoosh, the application of μ generates a new, merged tuple, but the old ones may stay. However, Swoosh applies a pruning process based on an abstract domination partial order between records, $\preceq^S$. The framework concentrates mostly on the *merge domination relation* $\leq$, which is defined by:

$$r_1 \leq r_2 \ :\Longleftrightarrow \ M(r_1,r_2) = \mathit{true} \text{ and } \mu(r_1,r_2) = r_2. \tag{11}$$

The $I^sC^sA^sR^s$ properties make $\leq$ a partial order with some pleasant and expected monotonicity properties [9].

According to Section 3, we may consider each of the partial orders $\preceq_{\{A_i\}}$ on the Dom_{A_i}: $s \preceq_{\{A_i\}} s' \ :\Leftrightarrow \ m_{\{A_i\}}(s,s') = s'$. They induce a $\preceq$ relation on Rec (cf. Definition 2).

Proposition 12. The general dominance relation $\preceq$ on Rec coincides with the merge domination relation $\leq$ obtained from M and μ.

Proof: For $\preceq_{\{A\}}$ on Dom_A it holds: $s \preceq_{\{A\}} s' \ :\Leftrightarrow \ m_{\{A\}}(s,s') = s' \ \Leftrightarrow \ s \cup s' = s' \ \Leftrightarrow \ s \subseteq s'$. Now, for records $r_1 = R(s_1^1, \ldots, s_n^1), r_2 = R(s_1^2, \ldots, s_n^2)$, it holds $r_1 \preceq r_2 :\Leftrightarrow$ for every $i, s_i^1 \preceq_{\{A\}} s_i^2 \Leftrightarrow$ for every $i, s_i^1 \subseteq s_i^2$.

On the other side, from (11) we obtain that, for records $r_1 = R(s_1^1, \ldots, s_n^1), r_2 = R(s_1^2, \ldots, s_n^2)$, it holds: $r_1 \leq r_2 \Leftrightarrow M(r_1,r_2) = \mathit{true}$ and for every $i, s_i^1 \subseteq s_i^2$.

[7] We use the superscript s, for Swoosh, to distinguish them from the properties listed in Section 3.

Since the s_i^j are non-empty, the first condition on the RHS is implied by the second one. ■

Given a dirty instance D, it is a natural question to ask about the relationship between the clean instance D^m obtained under our approach, by enforcing the above MDs, and the *entity resolution* instance D^s obtained directly via Swoosh. The entity resolution D^s is defined in [9, Def. 2.3] through the conditions: 1. $D^s \subseteq \bar{D}$. 2. $\bar{D} \leq D^s$. 3. D^s is $\subseteq$-minimal for the two previous conditions. Here, the partial-order $\leq$ between instances is induced by the partial order $\leq$ between records as in Definition 2. Instance $\bar{D}$ is the *merge closure* of D, i.e., the $\subseteq$-minimal instance that includes D and is closed under M: $r_1, r_2 \in \bar{D}$ and $M(r_1, r_2) = \mathit{true} \ \Rightarrow \mu(r_1, r_2) \in \bar{D}$.

Notice that, in order to obtain D^m, tuple identifiers are introduced and kept, whereas under Swoosh, there are no tuple identifiers and new tuples are generated (via μ) and some are deleted (those $\leq$-dominated by other tuples). In consequence, the elements of D and D^m under the MD framework are of the form $R(t, s_1, \ldots, s_n)$, and those in D and D^s under Swoosh are the records r of the form $R(s_1, \ldots, s_n)$. Since t is a tuple identifier, for every $R(t, s_1, \ldots, s_n)$, $r(t)$ denotes the record $R(s_1, \ldots, s_n)$.

Proposition 13. (a) For every r in D^s there is a tuple in D^m with tuple identifier t, such that $r(t) = r$.
(b) For every tuple $t \in D^m$, there is a record $r \in D^s$, such that $r(t) \leq r$.

Proof: (sketch) As a preliminary and useful remark, let us mention that the $I^sC^sA^sR^s$ properties make $\leq$ a partial order with the following monotonicity properties [9]: (A) $M(r_1, r_2) = \mathit{true} \implies r_1 \leq \mu(r_1, r_2)$ and $r_2 \leq \mu(r_1, r_2)$. (B) $r_1 \leq r_2$ and $M(r_1, r) = \mathit{true} \implies M(r_2, r) = \mathit{true}$. (C) $r_1 \leq r_2$ and $M(r_1, r) = \mathit{true} \implies \mu(r_1, r) \leq \mu(r_2, r)$. (D) $r_1 \leq s,\ r_2 \leq s$ and $M(r_1, r_2) = \mathit{true} \implies \mu(r_1, r_2) \leq s$.

More specifically for our proof, first notice that every application of μ can be simulated by a finite sequence of enforcement of the MDs in (10). More precisely, given two tuples $R(t_1, \bar{s}^1), R(t_2, \bar{s}^2)$ in an instance D, such that $M(r(t_1), r(t_2))$ holds, then $\mu(r(t_1), r(t_2)) = r(t)$ for some tuple $R(t,\ r(t))$ of the form $m_{\{A_{i_1}\}} \cdots m_{\{A_{i_n}\}}(R(t_1, \bar{s}^1), R(t_2, \bar{s}^2))$, i.e., obtained by enforcing the MDs. Furthermore, it holds $r(t_1) \leq r(t)$ and $r(t_2) \leq r(t)$.

Conversely, every enforcement of an MD in (10) is dominated by a tuple obtained through the application of μ. More precisely, for tuples $R(t_1, \bar{s}^1), R(t_2, \bar{s}^2)$ in an instance D for which $s_j^1 \approx_{\{A_j\}} s_j^2$ holds, it also holds $M(t_1(r), t_2(r))$, and $m_{\{A_j\}}(R(t_1, \bar{s}^1), R(t_2, \bar{s}^2)) \leq R(t, \mu(t_1(r), t_2(r))$ for some tuple id t (actually, t_1 or t_2).

Now, for (a), consider $D^m\!\downarrow := \{r(t) \mid R(t, \bar{s}) \in D^m\}$ (from where duplicates are eliminated). It is good enough to prove that $D^s \subseteq D^m\!\downarrow$. For this it suffices to prove that $D^m\!\downarrow$ satisfies conditions 1. and 2. on the entity resolution instance, namely: 1. $D^m\!\downarrow \subseteq \bar{D}$ and 2. $\bar{D} \leq D^m\!\downarrow$. The first condition follows from the definition (or construction) of D^m as a stable instance obtained by minimally applying the MDs and when justified only. The second condition follows from the simulation and properties of μ as a finitely long enforcement of the MDs.

Now (b) follows from the domination of a tuple obtained by applying one MD by a tuple obtained applying μ as described above. ■

This result shows that in the special case of Swoosh, where the merge function takes the union of two attribute value sets, the clean instance resulting from of our chase procedure with matching dependencies is equivalent to the Swoosh entity resolution (more precisely, they are equivalent if we look at the reduced version of the clean instance). This is an interesting special case of Corollary 4, where the least upper bound of clean instances is dominated by Swoosh entity resolution, and the reverse *does* hold.

9 Discussion

9.1 Associativity of matching functions

Associativity of a matching function is a natural assumption, not only because without it we cannot have a lattice and a terminating chase, etc., but also because it is an intuitive requirement in any entity resolution process such as ours. That is, when during the process we identify three or more data values that are representing the same entity, the result of collapsing them into one value should not depend on the order in which we visit those values.

We have made the assumption of associativity, and have developed our theoretical framework under it. In particular, associativity is crucial for finite termination (cf. Example 14 below). It could be interesting to do something similar without that assumption (but possibly with other assumptions).

Example 14. Consider the schema $Salary(name, amount)$ and the matching dependency $Salary[name] \approx Salary[name] \rightarrow Salary[amount] \rightleftharpoons Salary[amount]$, Assume that the matching function is defined by $m_{amount}(n_1, n_2) := Avg(n_1, n_2)$.

Starting from the instance D_0 on the right-hand side, different computations are possible, depending on the order in which the MD is applied.

Salary	*name*	*amount*
	J. Doe	5000
	J. Doe	3000
	J. Doe	2000

The following is a possible computation:

Salary	*name*	*amount*
	J. Doe	$\underline{5000}$
	J. Doe	$\underline{3000}$
	J. Doe	2000

$\mapsto$

Salary	*name*	*amount*
	J. Doe	4000
	J. Doe	$\underline{4000}$
	J. Doe	$\underline{2000}$

$\mapsto$

Salary	*name*	*amount*
	J. Doe	$\underline{4000}$
	J. Doe	3000
	J. Doe	$\underline{3000}$

$\ldots$

The underlined values indicating the tuples chosen for matching. An infinite (but converging) computation is created in this case. ■

Still in our setting, if a matching function is not associative, e.g. if it takes the average of two numbers, we can always use the union of values and apply the aggregate function at the end. The details deserve further investigation. (Cf. [18, sec. 4.1] for a related discussion around the "union class".)

Example 15. (example 14 continued) If instead of applying m_{amount} as above, we apply a new matching function defined by $m'_{amount}(n_1, n_2) := \{n_1, n_2\}$, we obtain a clean instance after a finite computation:

Salary	*name*	*amount*
	J. Doe	5000
	J. Doe	3000
	J. Doe	2000

$\mapsto$

Salary	*name*	*amount*
	J. Doe	{5000, 3000}
	J. Doe	{5000, 3000}
	J. Doe	2000

$\mapsto$

Salary	*name*	*amount*
	J. Doe	{5000, 3000}
	J. Doe	{5000, 3000, 2000}
	J. Doe	{5000, 3000, 2000}

$\mapsto$

Salary	*name*	*amount*
	J. Doe	{5000, 3000,2000}
	J. Doe	{5000, 3000, 2000}
	J. Doe	{5000, 3000, 2000}

Now, if we are interested in average as an aggregate function, we can apply it to the set-value in common, namely {5000, 3000, 2000}, obtaining the instance on the right-hand side.

Salary	*name*	*amount*
	J. Doe	3333.3
	J. Doe	3333.3
	J. Doe	3333.3

■

The next example shows a non-convergent behavior in the absence of associativity.

Example 16. Consider the schema $R(A, B)$ and the matching dependency φ : $R[A] \approx R[A] \rightarrow R[B] \rightleftharpoons R[B]$. In the following instance D_0, assume that $a_1 \approx a_2$ and $a_2 \approx a_3$. Furthermore, let m_B be an idempotent, commutative, and non-associative matching function, partially defined as follows:

$$
\begin{array}{ll}
m_B(b_1, b_2) = m_B(b_2, b_1) = b_4, & m_B(b_3, b_4) = m_B(b_4, b_3) = b_2, \\
m_B(b_2, b_4) = m_B(b_4, b_2) = b_1, & m_B(b_1, b_4) = m_B(b_4, b_1) = b_3, \\
m_B(b_2, b_3) = m_B(b_3, b_2) = b_1. &
\end{array}
$$

Observe that the chase sequence that starts from D_0 and alternates between enforcing φ on the first two tuples and the last two tuples gets into a non-terminating loop (D_6 is the same as D_3).

D_0	A	B
	a_1	b_1
	a_2	b_2
	a_3	b_3

$\mapsto$

D_1	A	B
	a_1	b_4
	a_2	b_4
	a_3	b_3

$\mapsto$

D_2	A	B
	a_1	b_4
	a_2	b_2
	a_3	b_2

$\mapsto$

D_3	A	B
	a_1	b_1
	a_2	b_1
	a_3	b_2

$\mapsto$

D_4	A	B
	a_1	b_1
	a_2	b_4
	a_3	b_4

$\mapsto$

D_5	A	B
	a_1	b_3
	a_2	b_3
	a_3	b_4

$\mapsto$

D_6	A	B
	a_1	b_3
	a_2	b_2
	a_3	b_2

$\mapsto$

D_6	A	B
	a_1	b_1
	a_2	b_1
	a_3	b_2

$\mapsto \ldots$

■

9.2 Data management with partially ordered domains

The domination-monotone relational query language introduced uses the lattice-theoretic structure of the domains, which is interesting in its own right. It certainly deserves further investigation, independently from data cleaning under matching dependencies.

It is interesting to explore its connections with querying databases over partially ordered domains, with incomplete or partial information [38, 29, 34, 33], with query relaxation in general [32, 23], and with relational languages based on similarity relations [30].

9.3 Logic programs for data cleaning under MDs

The class of repairs of an inconsistent database (w.r.t. integrity constraints) [10, 11] has been specified by means of logic programs with the stable model semantics. That is, the repairs are represented by, and computed as, the stable models of the logic program. In consistent query answering this approach has led to useful insights and implementations [6, 26, 16, 14]. In particular, consistent answers to queries can be obtained by cautiously reasoning from the program.

We are currently investigating the use of logic programs with stable model semantics for the specification of clean instances, and for doing clean query answering. In particular, the programs can be used to provide declarative versions of the Swoosh algorithms.

9.4 Related work

As indicated above, much work has been done around entity resolution (data fusion, record linkage, etc.) [17, 12], and much of that work has concentrated on algorithms and measures for duplicate detection [37]. In our work we have not considered detection. Rather, we abstract away duplicate detection by means of the similarity relations.

Matching dependencies are introduced in [19, 20], which provide the basis of our work. Their approach is both *generic*, in the sense that different ways of capturing the similarities between data items and of matching them can be accommodated in that framework. It is also *declarative* in the sense that the results of the matching processes are specified by means of logical formulas, and not by means of an algorithm (generic or ad hoc). Actually, the declarative specification could be implemented in different ways. We enriched this framework by introducing matching functions, which are still generic, and the specification is still declarative.

The Swoosh methodology for entity resolution [9] is also generic, but not declarative, in the sense that the semantics of the system is not captured in terms of models of a logical specification of the instances resulting from the cleaning process.[8] Several algorithms are presented for different cases. One of them, instead of working at the full record level (cf. Section 8.2), considers doing the matching on the basis of values for *features*, which, consider certain combinations of attributes [9, sec. 4]. This is in some sense close to the spirit of MDs. However, since the semantics of features is not fully developed, it is difficult to make a precise comparison. The authors of [9] acknowledge inspiration by the generic and declarative aspects of [28] and [24], resp.

Swoosh has been extended in [18] with *negative rules*. They are used to avoid inconsistencies (e.g. w.r.t. semantic constraints) that could be introduced by indiscriminate matching. From this point of view, certain elements of *database repairs* [10] are introduced into the picture (cf. [18, sec. 2.4]). In this direction, the combination of database repairing and MDs is studied in [21].

A declarative framework for collective entity matching of large data sets using domain-specific soft and hard constraints is proposed in [4]. The constraints specify the matchings. They use a novel Datalog style language, *Dedupalog*, to write the constraints as rules. The matching process tries to satisfy all the hard constraints, but minimizing the number of violations to the soft constraints. Dedupalog is used for identifying groups of tuples that could be merged. They do not do the merging or base their work on MDs.

Another declarative approach to ER is presented in [39]. The emphasis is placed mainly on the detection of duplicates rather than on the actual merging. An ontology expressed in a logical language based on RDF-S, OWL-DL and SWRL [2] is used for this task. Reconciliation rules are captured by SWRL. Also negative rules that prevent reconciliation of certain values can be expressed, much in the spirit of Swoosh with negative rules [18].

A treatment of entity resolution via matching dependencies that does not use matching matching functions, but a *minimal number of arbitrary changes* to do the matchings is presented in [25]. A semantics for clean instances and a corresponding chase procedure are proposed. Some connections with database repairs and consistent query answering are established.

10 Conclusions

The introduction of matching dependencies (MDs) in [19] has been a valuable addition to data quality and data cleaning research. They can be regarded as *data quality constraints* that are declarative in nature and are based on a precise model-theoretic semantics. They are bound to play an important role in database research and practice, together (and in combination) with classical integrity constraints.

In this work we have made several contributions to the semantics of matching dependencies. We have refined the original semantics introduced in [20], addressing some important open issues, but we have also introduced into the semantic

[8] In our MD framework the sets of MDs provide a logical specification, and the semantics is model-theoretic, as captured by the clean instances. Admittedly, the latter have a procedural component.

framework the notion of *matching function*. For entity resolution we need to know and spell out *how* attribute values have to be merged or identified, a key piece missing from the proposal of [20]. Matching functions fill this void.

The matching functions, under certain natural assumptions, induce lattice-theoretic structures in the attribute domains. This led us to introduce a partial order of domination between instances, and allowed us to compare them in terms of information content. The same domination order was then applied to sets of query answers. We also investigated the interaction of matching functions with similarity relations in the attribute domains.

On the basis of all these notions, we defined the class of clean instances for a given dirty instance. They are the intended and admissible instances that could be obtained after enforcing the matching dependencies. The clean instances were defined by means of a chase-like procedure that enforces the MDs, while not making unjustified changes on other attribute values, thereby capturing an essence of "minimality" of changes. W.r.t. the "lens" of domination order, the chase procedure improves the information content stepwise.

The notion of clean answer to a query posed to the dirty database was defined as a pair formed by a lower and an upper bound in terms of information content for the query answers. In this context we studied the notion of monotone query w.r.t. the domination order and how to relax a query into a monotone one that provides more informative answer than the original one.

We addressed some problems around the enforcement of a set of matching dependencies for purposes of data cleaning based on the original proposal of [19, 20], by explicitly making use of matching functions. We studied issues such as the existence and uniqueness of clean instances, the computational cost of computing them, and the complexity of computing clean answers. We identified cases where clean query answering is tractable, e.g., when there is a single clean instance. However, we established that this problem is intractable in general. We proposed polynomial time approximations. The assessment of these approximations and experimentation with them are part of our ongoing research, which also includes identifying other tractable cases, and developing efficient and more accurate approximations to the intractable cases.

Acknowledgments. This work was supported by NSERC Strategic Network on Business Intelligence (BIN ADC01, Years 1 and 2) and (BIN ADC05, Year 3); and NSERC/IBM CRDPJ/371084-2008, which is gratefully acknowledged. L. Bertossi is a Faculty Fellow of the IBM Center for Advanced Studies.

References

[1] Abiteboul, S., Kanellakis, P.C., Grahne, G. On the Representation and Querying of Sets of Possible Worlds. *Theoretical Computer Science*, 1991, 78(1):158-187.

[2] Antoniou, G. and van Harmelen, F. *A Semantic Web Primer*, 2nd ed. The MIT Press, 2008.

[3] Afrati, F. and Kolaities, Ph. Answering Aggregate Queries in Data Exchange. In Proc. ACM PODS, 2008, pp. 129-138.
[4] Arasu, A., Re, Ch. and Suciu, D. Large-Scale Deduplication with Constraints Using Dedupalog. In Proc. ICDE, 2009, pp. 952-963.
[5] Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. In Proc. ACM PODS, 1999, pp. 68-79.
[6] Arenas, M., Bertossi, L. and Chomicki, J. Answer Sets for Consistent Query Answering in Inconsistent Databases. *Theory and Practice of Logic Programming*, 2003, 3(4-5):393-424.
[7] Arenas, M., Bertossi, L., Chomicki, J., He, X., Raghavan, V. and Spinrad, J. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science*, 2003, 296(3):405-434.
[8] Bancilhon, F. and Khoshafian, S. A Calculus for Complex Objects. *Journal of Computer ans Systems Sciences*, 1989, 38(2):326-340.
[9] Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Euijong Whang, S. and Widom, J. Swoosh: A Generic Approach to Entity Resolution. *VLDB Journal*, 2009, 18(1):255-276.
[10] Bertossi, L. Consistent Query Answering in Databases. *ACM Sigmod Record*, 2006, 35(2):68-76.
[11] Bertossi, L. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers. Synthesis Lectures on Data Management, 2011.
[12] Bleiholder, J. and Naumann, F. Data Fusion. *ACM Computing Surveys*, 2008, 41(1).
[13] Buneman, P., Jung, A. and Ohori, A. Using Powerdomains to Generalize Relational Databases. *Theoretical Computer Science*, 1991, 91(1):23-55.
[14] Caniupan, M. and Bertossi, L. The Consistency Extractor System: Answer Set Programs for Consistent Query Answering in Databases. *Data & Knowledge Engineering*, 2010, 69(6):545-572.
[15] Chomicki, J. Consistent Query Answering: Five Easy Pieces. In Proc. ICDT, 2007, Springer LNCS 4353, pp. 1-17.
[16] Eiter, T., Fink, M., Greco, G. and Lembo, D. Repair Localization for Query Answering from Inconsistent Databases. *ACM Transactions on Database Systems*, 2008, 33(2).
[17] Elmagarmid, A., Ipeirotis, P. and Verykios, V. Duplicate Record Detection: A Survey. *IEEE Transactions in Knowledge and Data Engineering*, 2007, 19(1):1-16.
[18] Whang, S.E., Benjelloun, O. and Garcia-Molina, H. Generic Entity Resolution with Negative Rules. *VLDB Journal*, 2009, 18(6):1261-1277.
[19] Fan, W. Dependencies Revisited for Improving Data Quality. In Proc. ACM PODS, 2008, 159-170.
[20] Fan, W., Jia, X., Li, J. and Ma, S. Reasoning about Record Matching Rules. In Proc. VLDB, 2009, 2(1):407-418.
[21] Fan, W., Li, J., Ma, Sh., Tang, N. and Yu, W.. Interaction between Record Matching and Data Repairing. In Proc. ACM SIGMOD, 2011, pp. 469-480.
[22] Fuxman, A., Fazli, E. and Miller, R. ConQuer: Efficient Management of Inconsistent Databases. In Proc. ACM SIGMOD, 2005, pp. 155-166.
[23] Gaasterland, T., Godfrey, P. and Minker, J. Relaxation as a Platform for Cooperative Answering. *J. Intelligent Information Systems*, 1992, 1(3/4):293-321.
[24] Galhardas, H., Florescu, D., Shasha, D., Simon, E. and Saita, C-A.. Declarative Data Cleaning: Language, Model, and Algorithms. In Proc. VLDB, 2001, pp. 371-380.

[25] Gardezi, J., Bertossi, L. and Kiringa, I. Matching Dependencies with Arbitrary Attribute Values: Semantics, Query Answering and Integrity Constraints". Proc. of the International Workshop on Logic in Databases (LID'11), ACM Press, 2011.

[26] Greco, G., Greco, S. and Zumpano, E. A Logical Framework for Querying and Repairing Inconsistent Databases. *IEEE Transactions on Knowledge and Data Engineering*, 2003, 15(6):1389-1408.

[27] Gunter, C.A. and Scott, D.S. Semantic Domains. Chapter 12 in *Handbook of Theoretical Computer Science*, Vol. B, Elsevier, 1990.

[28] Hernndez, M. and Stolfo, S. The Merge/Purge Problem for Large Databases. In Proc. ACM SIGMOD, 1995, pp. 127-138.

[29] Imielinski, T. and Lipski Jr., W. Incomplete Information in Relational Databases. *Journal of the ACM*, 1984, 31(4):761-791.

[30] Jagadish, H., Mendelzon, A., and Milo, T. Similarity-Based Queries. Proc. ACM PODS, 1995, pp. 36-45.

[31] Kifer, M. and Lausen, G. F-Logic: A Higher-Order language for Reasoning about Objects, Inheritance, and Scheme. In Proc. ACM SIGMOD, 1989, pp. 134-146.

[32] Koudas, N., Li, Ch., Tung, A. and Vernica. R. Relaxing Join and Selection Queries. In Proc. VLDB, 2006, pp. 199-210.

[33] Levene, M. and Loizou, G. Database Design of Incomplete Relations. *ACM Transactions on Database Systems*, 1999, 24:35-68.

[34] Libkin, L. A Semantics-Based Approach to Design of Query Languages for Partial Information. In *Semantics in Databases*, 1998, Springer LNCS 1358, pp. 170-208.

[35] Libkin, L. Data Exchange and Incomplete Information. In Proc. ACM PODS, 2006, pp. 60-69.

[36] Lipski Jr., W. On Semantic Issues Connected with Incomplete Information Databases. *ACM Transactions on Database Systems*, 1979, 4(3):262-296.

[37] Naumann, F. and Herschel, M. *An Introduction to Duplicate Detection.* Morgan & Claypool Publishers. Synthesis Lectures on Data Management, 2010.

[38] Ng, W., Levene, M. and Fenner, T. On the Expressive Power of the Relational Algebra with Partially Ordered Domains. *International Journal of Computer Mathematics*, 2000, 71:53-62.

[39] Saïs, F., Pernelle, N. and Rousset, M.-Ch. L2R: A Logical Method for Reference Reconciliation. In Proc. AAAI, 2007, pp. 329-334.