On Chase Variations with Matching Dependencies for Entity Resolution

(DRAFT)[⋆]

Leopoldo Bertossi and **Jaffer Gardezi**

1 Introduction

Given a set of matching dependencies (MDs), the semantics of *resolved instances* is given in [2] in terms of a chase procedure. Those instances are obtained after a finite number of steps, at each of which the whole set of MDs is enforced, and are also *stable*, i.e. no more MD enforcement are possible or necessary. However, the semantics in [2] has the drawback that not all chase sequences terminate (see [2, Example 7]).

In this work we investigate in more detail the properties of the chase used to define the MD-semantics introduced in [2], and also some possible variations of the chase that lead to similar or different results.

2 Preliminaries

We consider a relational schema $\mathcal{S}$ that includes an enumerable, possibly infinite domain U, and a finite set $\mathcal{R}$ of database predicates. Elements of U are represented by lower case letters near the beginning of the alphabet. $\mathcal{S}$ determines a first-order (FO) language $L(\mathcal{S})$. An instance D for $\mathcal{S}$ is a finite set of ground atoms of the form $R(\bar{a})$, with $R \in \mathcal{R}$, say of arity n, and $\bar{a} \in U^n$. $R(D)$ denotes the extension of R in D. Every predicate $R \in \mathcal{S}$ has a set of attribute, denoted $attr(R)$. As usual, we sometimes refer to attribute A of R by $R[A]$. We assume that all the attributes of a predicate are different, and that we can identify attributes with *positions* in predicates, e.g. $R[i]$, with $1 \leq i \leq n$. If the ith attribute of predicate R is A, for a tuple $t = (c_1, \ldots, c_n) \in R(D)$, $t^D_R[A]$ (usually, simply $t_R[A]$ or $t[A]$ if the instance is understood) denotes the value c_i. For a sequence $\bar{A}$ of attributes in $attr(R)$, $t[\bar{A}]$ denotes the tuple whose entries are the values of the attributes in $\bar{A}$. Attributes have and may share subdomains of U.

In the rest of this section, we summarize some of the assumptions, definitions, notation, and results from [1], that we will need.

We will assume that every relation in an instance has an auxiliary attribute, a surrogate key, holding values that act as *tuple identifiers*. Tuple identifiers are never created, destroyed or changed. They do not appear in MDs, and are used to identify different versions of the same original tuple that result from the matching process. We usually leave them implicit; and "tuple identifier attributes" are commonly left out when specifying a database schema. However, when explicitly represented, they will be the "first" attribute of the relation. For example, if $R \in \mathcal{R}$ is n-ary, $R(t, c_1, \ldots, c_n)$ is a tuple with

[⋆] April, 2014

id t, and is usually written as $R(t, \bar{c})$. We usually use the same symbol for a tuple's identifier as for the tuple itself. Tuple identifiers are unique over the entire instance.

Two instances over the same schema that share the same tuple identifiers are said to be *correlated*. In this case it is possible to unambiguously compare their tuples, and as a result, also the instances.

As expected, some of the attribute domains, say A, have a built-in binary similarity relation $\approx_A$. That is, $\approx_A \subseteq Dom(A) \times Dom(A)$. It is assumed to be reflexive and symmetric. Such a relation can be extended to finite lists of attributes (or domains therefor), componentwise. For single attributes or lists of them, the similarity relation is is generically denoted with $\approx$.

A *matching dependency* (MD), involving predicates R, S, is an expression (or rule), m, of the form

$$m: \ R[\bar{A}] \approx S[\bar{B}] \ \rightarrow \ R[\bar{C}] \doteq S[\bar{E}], \tag{1}$$

with $\bar{A} = (A_1, ..., A_k)$, $\bar{C} = (C_1, ..., C_{k'})$ lists of (different) attributes from $attr(R)$; and $\bar{B} = (B_1, ..., B_k)$, $\bar{E} = (E_1, ..., E_{k'})$ lists of attributes from $attr(S)$.[1]

The set of attributes on the left-hand-side (LHS) of the arrow in m is denoted with $LHS(m)$. Similarly for the right-hand-side (RHS).

In relation to (1), the attributes in a *corresponding pair* (A_i, B_i) or (C_i, E_i) are assumed to share a common domain; and in particular, a *similarity relation* $\approx_i$. In consequence, the condition on the LHS of (1) means that, for a pair of tuples t_1 in R and t_2 in S, $t_1[A_i] \approx_i t_2[B_i]$, $1 \leq i \leq k$. Similarly, the expression on the RHS means $t_1[A_i] \doteq t_2[B_i]$, $1 \leq i \leq k'$. Here, $\doteq$ means that the values should be updated to the same value.

Accordingly, the intended semantics of the MD in (1) is that, for an instance D, if any pair of tuples, $t_1 \in R(D)$ and $t_2 \in S(D)$, satisfy the similarity conditions on the LHS, then for the same tuples (or tuple ids), the attributes on the RHS have to take the same values, possibly through updates that may lead to a new version of D.

We assume that all sets $\mathcal{M}$ of MDs are in *standard form*, i.e. for no two different MDs $m_1, m_2 \in \mathcal{M}$, $LHS(m_1) = LHS(m_2)$. All sets of MDs can be put in this form. MDs in a set $\mathcal{M}$ can interact in the sense that a matching enforced by one of them may create new similarities that lead to the enforcement of another MD in $\mathcal{M}$. This intuition is captured through the *MD-graph*.

Definition 1. [2] Let $\mathcal{M}$ be a set of MDs in standard form. The *MD-graph* of $\mathcal{M}$, denoted $MDG(\mathcal{M})$, is a directed graph with a vertex m for each $m \in \mathcal{M}$, and an edge from m to m' iff $RHS(m) \cap LHS(m') \neq \emptyset$.[2] If $MDG(\mathcal{M})$ contains edges, $\mathcal{M}$ is called *interacting*. Otherwise, it is called *non-interacting* (NI). □

3 Matching Dependencies and Resolved Answers

Updates as prescribed by an MD m are not arbitrary. The updates based on m have to be justified by m, as captured through the notion of *modifiable value* in an instance.

[1] We assume that the MDs are defined in terms of the same schema $\mathcal{S}$.

[2] That is, they share at least one corresponding pair of attributes.

Definition 2. Let D be an instance, $\mathcal{M}$ a set of MDs, and $\mathcal{P}$ be a set of pairs (t, G), where t is a tuple of D and G is an attribute of t. (a) For a tuple $t_R \in R(D)$ and C an attribute of R, the value $t_R^D[C]$ is *modifiable wrt* $\mathcal{P}$ if there exist $S \in \mathcal{R}$, $t_S \in S(D)$, an $m \in \mathcal{M}$ of the form $R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \doteq S[\bar{E}]$, and a corresponding pair (C, E) of $(\bar{C}, \bar{E})$ in m, such that $(t_S, E) \in \mathcal{P}$ and one of the following holds:

1. $t_R[\bar{A}] \approx t_S[\bar{B}]$, but $t_R[C] \neq t_S[E]$.
2. $t_R[\bar{A}] \approx t_S[\bar{B}]$ and $t_S[E]$ is modifiable wrt $\mathcal{P} \setminus \{(t_S, E)\}$.

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

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

Example 1. Consider $m: R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$ on schema $R[A, B]$, and the instance $R(D)$ shown below. Assume that the only non-trivial similarities are $a_1 \approx a_2 \approx a_3$ and $b_1 \approx b_2$. Since $a_2 \approx a_3$ and $c_1 \neq c_3$, $t_2[B]$ and $t_3[B]$ are modifiable (base case). With case 2 of Definition 2, since $a_1 \approx a_2$, and $t_2[B]$ is modifiable, we obtain that $t_1[B]$ is modifiable. For $t_5[B]$ to be modifiable, it must be modifiable wrt $\{(t_i, B) \mid 1 \leq i \leq 4\}$, and via t_4. According to case 2 of Definition 2, this requires $t_4[B]$ to be modifiable wrt $\{(t_i, B) \mid 1 \leq i \leq 3\}$. However, this is not the case since there is no t_i, $1 \leq i \leq 3$, such that $t_4[A] \approx t_i[A]$. Therefore $t_5[B]$ is not modifiable. A symmetric argument shows that $t_4[B]$ is not modifiable. □

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

Definition 3. [1] Let D, D' be correlated instances, and $\mathcal{M}$ a set of MDs. (D, D') *satisfies* $\mathcal{M}$ (modulo unmodifiability), denoted $(D, D') \vDash_{um} \mathcal{M}$, iff: 1. For any pair of tuples $t_R \in R(D)$, $t_S \in S(D)$, if there exists an $m \in \mathcal{M}$ of the form $R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \doteq S[\bar{E}]$ and $t_R[\bar{A}] \approx t_S[\bar{B}]$, then for the corresponding tuples (i.e. with same ids) $t'_R \in R(D')$ and $t'_S \in S(D')$, it holds $t'_R[\bar{C}] = t'_S[\bar{E}]$. 2. For any tuple $t_R \in R(D)$ and any attribute G of R, if $t_R[G]$ is *non-modifiable*, then $t'_R[G] = t_R[G]$. □

Intuitively, D' in Definition 3 is a new version of D that is produced after a single update. Since the update involves matching values (i.e. making them equal), it may produce "duplicate" tuples, i.e. that only differ in their tuple ids. They would possibly be merged into a single tuple in the a data cleaning process. However, we keep the two versions. In particular, D and D' have the same number of tuples. Keeping or eliminating duplicates will not make any important difference in the sense that, given that tuple ids are never updated, two duplicates will evolve in exactly the same way as subsequent updates are performed. Duplicate tuples will never be subsequently "unmerged".

This definition of MD satisfaction does not require that updates preserve similarities. Similarity preservation may force undesirable changes [1]. The existence of the

updated instance D' for D is guaranteed [1]. Furthermore, our definition does not allow unnecessary changes from D to D'. Definitions 2 and 3 imply that only values of attributes that appear to the right of the arrow in some MD are subject to updates. Hence, they are called *changeable attributes*.[3]

Definition 3 allows us to define a *clean instance* wrt $\mathcal{M}$ as the result of a chase-like procedure, each step being satisfaction preserving.

Definition 4. [1] (a) A *resolved instance* (RI) for D wrt $\mathcal{M}$ is an instance D', such that there is a sequence of instances $D_0, D_1, ..., D_n$ such that: $0 \leq n$, $D_0 = D$, $D_n = D'$; and $(D_0, D_1) \models_{um} \mathcal{M}$, $(D_1, D_2) \models_{um} \mathcal{M}$,..., $(D_{n-1}, D_n) \models_{um} \mathcal{M}$, $(D_n, D_n) \models_{um} \mathcal{M}$, and $(D_i, D_i) \not\models_{um} \mathcal{M}$, for $i < n$. Because $(D', D') \models_{um} \mathcal{M}$, we say D' is *stable*. (b) D' is a *minimally resolved instance* (MRI) for D wrt $\mathcal{M}$ if it is a resolved instance and it minimizes the overall number of attribute value changes wrt D. (c) $MRI(D, \mathcal{M})$ denotes the class of MRIs of D wrt $\mathcal{M}$. □

Remark 1. The case $n = 0$ in Definition 4, is when and only when D is already resolved. □

Because of the minimality constraint, the range of possible update values tends to be more restricted for MRIs than for resolved instances. In fact, it may even be necessary to update to a value outside the active domain, as Example 2 shows.

For resolved instances, do we do not need to pick up constants from outside the active domain. That is, there are always resolved instances that use constants from the active domain only. For example, we could choose a value v from within the active domain and use it as the update value for all updates. The update process would eventually terminate with a resolved instance, since we are increasing the frequency of occurrence of v in the instance with each update.

For minimally resolved instances, there are cases when we must use a value from outside the active domain to obtain a minimally resolved instance. This happens when restricting to the active domain would result in accidental similarities, leading to further updates and more changes. (This leaves room for a concept of minimality relative to the active domain.)

Example 2. Consider relation $R[A, B, C]$, equality as the similarity relation, and the MDs and instance below:

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

$R(D)$	A	B	C
t_1	a	d	f
t_2	a	e	f
t_3	b	d	g
t_4	b	e	g
t_5	c	d	h
t_6	c	e	h

The domains of the attributes are disjoint. Suppose we update D according to the MDs, choosing the update values from the active domain. In the first update, each of the sets $\{t_1[B], t_2[B]\}$, $\{t_3[B], t_4[B]\}$, and $\{t_5[B], t_6[B]\}$ must be merged to either d or e.

[3] Not to be confused with "modifiability", that applies to tuples.

Thus, two of the sets must be updated to the same value. This implies that in the next (and final) update, there will be a set of four tuples whose values for the C attribute must be merged. Therefore, the total number of changes for the resulting resolved instance will be at least 5: 3 in the B column and 2 in the C column.

Now suppose we allow updates to values outside the active domain. In the first update, choose d and e as the update value for $\{t_1[B], t_2[B]\}$ and $\{t_3[B], t_4[B]\}$, respectively, and choose a value from outside the active domain for $\{t_5[B], t_6[B]\}$. The new B column values do not force any changes to the values in the C column, so the required number of changes for the resolved instance is the number of changes in the B column, which is 4. (We have ignored the fact that values in the C column will change in the first update, but it is easily verified that these changes can be undone in the second update.) Therefore, any minimal resolved instance must include a value from outside the active domain. □

There are cases of MDs and instances for which there are both non-terminating chase sequences, and also non-trivial (i.e. $n > 0$) terminating chase sequences. See Example 3.

It is possible to detect loops in the chase generation of resolved instances, as in the case of Example 3. That is, detect when the chase has produced an instance identical to one that was produced at an earlier step. This is because we could just compare the instance with all the previous instances.

Notice that not all non-terminating chases are cyclic (in the sense of Example 3), i.e. where the same instance is produced at more than one step of the chase. A non-terminating chase only requires that the sets of positions with similar values in the instance are the same as in a previously-generated instance. This implies that we are performing the same mergings of values over and over again, just with different values. This doesn't imply that any instance is repeated if the domain of attribute values is infinite.

Example 3. Consider relation $R[A, B]$, equality as the similarity relation, and the MDs and instance below:

$$m_1 : \ R[A] = R[A] \rightarrow R[B] \doteq R[B],$$
$$m_2 : \ R[B] = R[B] \rightarrow R[A] \doteq R[A],$$

$R(D)$	A	B
t_1	a	c
t_2	b	c
t_3	b	d
t_4	a	d

The chase may not terminate, which happens when the values oscillate, as in the following update sequence:

$R(D)$	A	B
t_1	a	c
t_2	b	c
t_3	b	d
t_4	a	d

$\mapsto$

$R(D)$	A	B
t_1	a	c
t_2	a	d
t_3	b	d
t_4	b	c

$\mapsto$

$R(D)$	A	B
t_1	a	c
t_2	b	c
t_3	b	d
t_4	a	d

$\mapsto \cdots$

However, there are non-trivial terminating chase sequences:

$$\sigma: \quad \begin{array}{|c|c|c|}\hline R(D) & A & B \\ \hline t_1 & a & c \\ t_2 & b & c \\ t_3 & b & d \\ t_4 & a & d \\ \hline \end{array} \quad \mapsto \quad \begin{array}{|c|c|c|}\hline R(D) & A & B \\ \hline t_1 & f & h \\ t_2 & f & g \\ t_3 & e & g \\ t_4 & e & h \\ \hline \end{array} \ ,$$

with e, f, g, and h arbitrary. After this, a stable instance can be obtained by updating all values in the A and B to the same value. □

Remark 2. The following hold for the notion of (minimally) resolved instance:

1. For some sets $\mathcal{M}$ of MDs and initial instances D, there are admissible but non-terminating chase sequences. An example using the cyclic set $\mathcal{M}$ is Example 3 (see also [2, example 7]). Those cases exhibit a cyclic behavior, in the sense that after a finite number of steps, the same unresolved instance is obtained.
2. For every set $\mathcal{M}$ of MDs and initial instance D, there is at least one resolved instance, and then also a minimally resolved instance [2, theorem 2]. See σ in Example 3.
3. A natural question is about necessary and sufficient syntactic conditions on the set $\mathcal{M}$ of MDs (or its graph $MDG(\mathcal{M})$ or a refined version of it) that ensure that all chase sequences terminate, independently from the initial instance D.
4. Notice that a refined version of the graph $MDG(\mathcal{M})$, the *augmented graph*, was introduced in [2, definition 13]. When it is acyclic, every chase sequence terminates in a number of steps that is bounded above by a number that depends of the augmented graph alone, not by the size of the initial instance. So, there should be room for an intermediate result about general finite termination, but in a number of steps that depends on the instance as well. This is an open problem.[4]

□

Example 4. Consider the MD $R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$ on predicate R, and the instance D. It has several resolved instances, among them, four that minimize the number of changes. One of them is D_1. A resolved instance that is not minimal in this sense is D_2.

$$\begin{array}{|c|c|c|}\hline R(D) & A & B \\ \hline t_1 & a_1 & c_1 \\ t_2 & a_1 & c_2 \\ t_3 & b_1 & c_3 \\ t_4 & b_1 & c_4 \\ \hline \end{array} \qquad \begin{array}{|c|c|c|}\hline R(D_1) & A & B \\ \hline t_1 & a_1 & c_1 \\ t_2 & a_1 & c_1 \\ t_3 & b_1 & c_3 \\ t_4 & b_1 & c_3 \\ \hline \end{array} \qquad \begin{array}{|c|c|c|}\hline R(D_2) & A & B \\ \hline t_1 & a_1 & c_1 \\ t_2 & a_1 & c_1 \\ t_3 & b_1 & c_1 \\ t_4 & b_1 & c_1 \\ \hline \end{array}$$

□

Since there is always an RI [1], there is always an MRI for an instance D wrt $\mathcal{M}$, even when there are non-terminating chase sequences.

In this work, as in [1, 2], we are investigating what we could call "the pure case" of MD-based entity resolution, which does not specify how the matchings are to be done,

[4] The potential source of undecidability is terminating sequences of unbounded length, since if there is a bound on terminating sequences, then all resolved instances could be computed by applying the chase until the bound is reached. We do not know of any example of unbounded sequences.

but only which values must be made equal. That is, the MDs have implicit existential quantifiers (for the values in common). The semantics we just introduced formally captures this pure case. We find situations like this in other areas of data management, e.g. with referential integrity constraints, tuple-generating dependencies in general, schema mappings in data exchange, etc.

The *resolved* answers to a query are *certain* for the class of MRIs for D wrt $\mathcal{M}$.

Definition 5. [1] Let $\mathcal{Q}(\bar{x})$ be a query expressed in the first-order language $L(\mathcal{S})$ associated to schema $\mathcal{S}$ of an instance D. A tuple of constants $\bar{a}$ from U is a *resolved answer* to $\mathcal{Q}(\bar{x})$ wrt the set $\mathcal{M}$ of MDs, denoted $D \models_{\mathcal{M}} \mathcal{Q}[\bar{a}]$, iff $D' \models \mathcal{Q}[\bar{a}]$, for every $D' \in MRI(D, \mathcal{M})$. We denote with $ResAn(D, \mathcal{Q}, \mathcal{M})$ the set of resolved answers to $\mathcal{Q}$ from D wrt $\mathcal{M}$. □

Example 5. (example 1 cont.) Since the only MRI for the original instance D is $R(D') = \{\langle t_1, a_1, c_1\rangle, \langle t_2, a_2, c_1\rangle, \langle t_3, a_3, c_1\rangle, \langle t_4, b_1, c_3\rangle, \langle t_5, b_2, c_3\rangle\}$, the resolved answers to the query $\mathcal{Q}(x, y)\colon\ R(x, y)$ are $\{\langle a_1, c_1\rangle, \langle a_2, c_1\rangle, \langle a_3, c_1\rangle, \langle b_1, c_3\rangle, \langle b_2, c_3\rangle\}$. □

For a query $\mathcal{Q}$ and set of MDs $\mathcal{M}$, the *resolved answer problem* is the problem of deciding, given a tuple $\bar{a}$ and instance D, whether or not $\bar{a} \in ResAn(D, \mathcal{Q}, \mathcal{M})$.

Remark 3.

1. The notion of resolver answer is non-trivial in the sense that it is always well-defined, i.e. for every class of MDs (and queries), in the sense that the class of minimally resolved instances is non-empty.
2. An open question is about the decidability of the *resolved answer problem* (RAP), i.e. membership of $\bar{a} \in ResAn(D, \mathcal{Q}, \mathcal{M})$.

Under certain conditions RAP is decidable, e.g. when the augmented graph is acyclic [2, theorem 5]. Also for certain classes of MDs with *special kinds of cycles*, e.g. the *hit-simple-cycle* (HSC), lead not only to a decidable resolved answer problem for certain conjunctive queries (*UJCQ* queries), but have a polynomial-time decision algorithm based on Datalog query rewriting [3, 4]. An example is the set in Example 8 below, for which, interestingly, there are also non-terminating chase sequences.

For HSC sets of MDs, RAP is decidable for every conjunctive query (*UJCQ* or not), because the resolved answers can be computed by intersecting the sets of answers obtained from the different resolved instances. This is possible, because it is possible to easily characterize, and compute, all the resolved instances on the basis of the stability requirement.[5] The chase sequences are bounded above by a number that depends on the MDs alone, not the initial instance [3, proposition 1]. However, RAP can be intractable for non-*UJCQ* queries [3, theorem 4].

4 Merge-Persistent Chase: Enforcing Termination

In this section we propose and investigate a modified chase-based semantics that guarantees that all chase sequences finitely terminate. Actually, in order to ensure chase

[5] Notice that for HSC sets there are non-terminating chase sequences (e.g. Example 3).

termination we will require that, whenever two duplicates are made equal by the chase, they cannot subsequently be made unequal. The intended clean instances for an initial instance D and a set $\mathcal{M}$ of MDs, will be those "final" instances that also have the property of stability (cf. Definition 6).

This non-separation requirement removes some inconsistencies that plague the chase as defined in the previous section. Specifically, it avoids making updates based on an assumption of equality of pairs of values that is subsequently withdrawn. This is illustrated in the example below.

Example 6. This example considers the following database instance and set of MDs, with similarity defined as equality:

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

$R(D)$	A	B	C	D
t_1	a	d	e	h
t_2	a	c	f	h
t_3	b	c	g	i

The following update sequence produces a minimally resolved instance D':

$R(D)$	A	B	C	D
t_1	a	d	e	h
t_2	a	c	f	h
t_3	b	c	g	i

$\mapsto$

$R(D_1)$	A	B	C	D
t_1	a	d	e	h
t_2	a	d	f	h
t_3	b	c	f	i

$\mapsto$

$R(D')$	A	B	C	D
t_1	a	d	e	h
t_2	a	d	e	h
t_3	b	c	f	h

Although this is a valid sequence of updates, the second update is inconsistent. In the second update, the duplicate values at positions $t_2[D]$ and $t_3[D]$ are merged on the assumption that the values at positions $t_2[C]$ and $t_3[C]$ are duplicates, but at the same time this assumption is withdrawn by setting $t_2[C]$ and $t_3[C]$ to different values.

A more consistent approach would be to choose the update values for the C column in the second update so that the identification of $t_2[C]$ and $t_3[C]$ in the first update is preserved. This leads to the resolved instance D'':

$R(D)$	A	B	C	D
t_1	a	d	e	h
t_2	a	c	f	h
t_3	b	c	g	i

$\mapsto$

$R(D_1)$	A	B	C	D
t_1	a	d	e	h
t_2	a	d	f	h
t_3	b	c	f	i

$\mapsto$

$R(D'')$	A	B	C	D
t_1	a	d	f	h
t_2	a	d	f	h
t_3	b	c	f	h

□

Although Definition 4 does not explicitly require that merged values not be unmerged, it is still true in that case that if the chase produces a pair of identical tuples t_1 and t_2 in a relation, none of the attributes in the two tuples can be unmerged at subsequent steps of the chase. This follows from the fact that, in the definition of MD satisfaction used in [2], all MDs are applied simultaneously to all tuples in the instance. More precisely, for any tuple t_3 and MD m, either both of the pairs (t_1, t_3) and (t_2, t_3)

satisfy the similarity condition of m or neither of them do. Therefore, if t_1 has its attribute values merged with those of t_3 through application of m, then those of t_2 must also be merged with those of t_3 (and t_2), and vice versa (see Examples 7 and 9).

Example 7. Consider relation $R[A, B, C]$, MDs

$$m_1 : \ R[A] \approx R[A] \rightarrow R[B] \doteq R[B],$$
$$m_2 : \ R[B] \approx R[B] \rightarrow R[C] \doteq R[C],$$

and the instance

$R(D)$	A	B	C
t_1	a	c_1	e
t_2	b	c_2	f
t_3	b	c_2	f

where the only similarity between unequal values is $c_1 \approx c_2$. The tuples t_2 and t_3 are identical (except for their tuple ids, which do not appear in MDs). Applying m_1 to D cannot unmerge $t_2[B]$ and $t_3[B]$, because $t_2[A]$ and $t_3[A]$ are both dissimilar to $t_1[A]$, so neither of $t_2[B]$ and $t_3[B]$ change. Applying m_2 to D cannot unmerge $t_2[C]$ and $t_3[C]$, because $t_2[B]$ and $t_3[B]$ are both similar to $t_1[B]$, so if they change, they both change to the same value. Notice that, in demonstrating that t_2 and t_3 cannot be unmerged, we are not using any additional restrictions on updates beyond those implied by the definition of MD satisfaction (Definition 3 in [2]). □

4.1 MP-resolved instances

The following is a modified version of Definition 4 of resolved instance. It captures the idea of disallowing the unmerging of values. It is based on a *merge-persistent chase* (MP-chase).

Definition 6. [1] (a) A *resolved instance* (RI) for D wrt $\mathcal{M}$ is an instance D', such that there is a sequence of instances $D_0, D_1, ..., D_n$ such that: (a) $0 \leq n$, $D_0 = D$, $D_n = D'$, and $(D_0, D_1) \models_{um} \mathcal{M}$, $(D_1, D_2) \models_{um} \mathcal{M}$,..., $(D_{n-1}, D_n) \models_{um} \mathcal{M}$, $(D_n, D_n) \models_{um} \mathcal{M}$, and $(D_i, D_i) \not\models_{um} \mathcal{M}$, for $i < n$; and (b) For any MD $R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \doteq S[\bar{E}]$ in $\mathcal{M}$ and any pair of tuples t and t', if $t[\bar{A}] \approx t'[\bar{B}]$ in D_i, then $t[\bar{C}] = t'[\bar{E}]$ in all D_j, $j > i$. □

According to this definition, the two merged attribute values must have the same value from some point on, but they could change the value in common. Example 8 below shows that two merged values may have to jump together to a new value in common.

This definition appeals to a chase procedure subject to the condition in (b), which makes us call it a *merging-persistent chase* (MP-chase). At each step we pass from an instance, say D_i, that does not satisfy the MDs, i.e. $(D_i, D_i) \not\models \mathcal{M}$, to an instance D_{i+1} such that $(D_i, D_{i+1}) \models_{um} \mathcal{M}$.

Definition 7. Let D be a database instance and $\mathcal{M}$ a set of MDs. The MP-chase *succeeds* if it finitely terminates with a stable instance (in which case no further MD enforcements are necessary). The MP-chase *fails* if it finitely terminates with a non-stable instance D_i for which there is no instance D_{i+1} with $(D_i, D_{i+1}) \models_{um} \mathcal{M}$. In either case we say that the MP-chase *terminates*. □

The set of chase sequences that can be produced with the new semantics is contained in the set of chase sequences under the semantics in Definition 4, because the new definition of chase adds a restriction to the previous one. The containment is strict when there are sequences of updates that lead to a non-stable instance such that no update can be made to the instance without violating the new restriction. This is illustrated in Example 9. The sets of chase sequences are the same for the above example, because there are essentially only two tuples, so there is no way to unmerge values in any case (this requires at least three tuples).

Theorem 1. The MP-chase always terminates after a number of steps that is polynomial in the number n of values in the instance.

Proof (sketch): The number of pairs of values in the instance is $O(n^2)$. Since the number of pairs of equal values grows at each step, and no MDs can be applied if all values are equal, there can be at most $O(n^2)$ steps. □

Example 8. (example 3 continued) Consider the same MDs and initial instance. The non-terminating chase sequences based on Definition 4 we had before are not admissible for Definition 6. However, any MP-chase sequence starting with instance D must terminate after at most two updates, as was shown with sequence σ at the end of Example 3 (σ is an MP-chase sequence). After the first update in σ, the only way the merged values remain equal after the second update is if all values in the A and B are updated to the same value. □

Although all MP-chase sequences terminate, they need not terminate in a clean instance, i.e. they may fail. This is illustrated in the next example.

Example 9. Consider relation $R[A, B, C]$, MDs

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

with equality as the similarity relation, and the instance

$R(D)$	A	B	C
t_1	a	c	$.e$
t_2	b	c	f
t_3	b	d	g
t_4	k	d	h

Consider the following sequence of updates.

$R(D)$	A	B	C
t_1	a	c	e
t_2	b	c	f
t_3	b	d	g
t_4	k	d	h

$\rightarrow$

$R(D')$	A	B	C
t_1	a	c	f
t_2	b	p	f
t_3	b	p	h
t_4	k	d	h

D' is not a clean instance, since it is not stable ($t_2[B] = t_3[B]$ but $t_2[C] \neq t_3[C]$). Also, by Definition 6, there is no valid update that can be made to D', since updating $t_2[C]$ and $t_3[C]$ to make them equal would produce an instance for which either $t_1[C] \neq t_2[C]$ or $t_3[C] \neq t_4[C]$, and all other values in D' are unmodifiable. Therefore, the chase terminates. Because it does not terminate in a clean instance, the chase fails.

Notice that the following instance D'' cannot be obtained by means of the MP-chase applied to D'.

$R(D'')$	A	B	C
t_1	a	c	h
t_2	b	p	h
t_3	b	p	h
t_4	k	d	h

This is because it modifies $t_1[C]$, which is not a modifiable value (the value of $t_1[B]$ is not equal to any of the other values in the B column.

Notice that, because t_1 and t_2 differ on some attributes in D', it is possible, under the semantics of [2], to unmerge the previously merged values $t_1[C]$ and $t_2[C]$ (and similarly for t_3 and t_4). More specifically, because $t_1[B] \neq t_3[B]$ but $t_2[B] = t_3[B]$, it is possible for $t_1[C]$ and $t_2[C]$ to become unequal when m_2 is applied to D'. For example, under the semantics of [2], the above chase sequence can be extended to produce a stable instance D'' as follows:

$R(D)$	A	B	C
t_1	a	c	e
t_2	b	c	f
t_3	b	d	g
t_4	k	d	h

$\rightarrow$

$R(D')$	A	B	C
t_1	a	c	f
t_2	b	p	f
t_3	b	p	h
t_4	k	d	h

$\rightarrow$

$R(D'')$	A	B	C
t_1	a	c	f
t_2	b	p	f
t_3	b	p	f
t_4	k	d	h

Unlike the case of identical tuples in Example 7, the additional restriction on updates in Definition 6 is required to prevent this unmerging of values. □

Although chase sequences can fail, it is still true that there exists a resolved instance (as in Definition 4) for every instance and set of MDs. Intuitively, this is because a single legal update can always be made choosing the same update value for all values of attributes with the same domain, i.e. a one-step chase [2, theorem 2]. The proof of existence of MP-resolved instances is similar, actually the same, because the resolved instance constructed in the proof in [2] is an MP-resolved instance.

Proposition 1. For an instance D subject to a set $\mathcal{M}$ of MDs, there is always an MP-resolved instance for D wrt $\mathcal{M}$.

Proof (sketch): For each domain of values of D, choose a value from the domain. At each step of the chase, when a value from a given domain needs to be updated, we update it to the value chosen for that domain. The only way the chase can fail is if two

values cannot be merged without undoing a previous merge. This can only happen if both of the values were previously merged with other values, and were updated to different values. By choice of update value, this is not possible. □

Example 10. (example 9 continued) An MP-resolved instance for the setting in Example 9 can be obtained by choosing the same value for all updates to the values in the C column.

For example, if D is updated in the same way as in Example 9 except that f is chosen as the update value for the C column, the following clean instance is obtained.

$R(D')$	A	B	C
t_1	a	c	f
t_2	b	p	f
t_3	b	p	f
t_4	k	d	f

□

Because Definition 6 is Definition 4 with a restriction, the set of resolved instances under Definition 6 is contained in the set of resolved instances under Definition 4.

Proposition 2. The set $mpRes(D, \mathcal{M})$ of MP-resolved instances is contained in the set $Res(D, \mathcal{M})$ of resolved instances as defined by Definition 4. □

The following example shows that the containment is strict.

Example 11. Consider again the schema and MDs of Example 9, and let D be the following instance.

$R(D)$	A	B	C
t_1	a	c	e
t_2	b	c	f
t_3	b	d	g
t_4	a	d	h

It is easy to see that, under the semantics of Definition 6, any resolved instance must have all values in the C column equal. However, the following chase is allowed under Definition 4.

$R(D)$	A	B	C		$R(D')$	A	B	C		$R(D'')$	A	B	C
t_1	a	c	e		t_1	a	c	f		t_1	a	c	f
t_2	b	c	f	$\mapsto$	t_2	b	p	f	$\mapsto$	t_2	b	p	h
t_3	b	d	g		t_3	b	p	h		t_3	b	p	h
t_4	a	d	h		t_4	a	c	h		t_4	a	c	f

□

4.2 MP-Resolved Query Answers

Minimally MP-resolved instances (MMPRIs) and *MP-resolved answers* can be defined by replacing "resolved" with "MP-resolved" in Definitions 4 (b) and 5, respectively. In this subsection, we discuss the problem of computing the MP-resolved answers.

It is currently an open problem to derive an upper bound for the complexity of the resolved answer problem. However, Theorem 1 implies that the analogous problem for MP-resolved answers (the *MP-resolved answer problem*) is in Π_2^P. Furthermore, we will now show that, for some sets of MDs, the MP-resolved answers to a query can be retrieved efficiently even though the resolved answer problem is intractable.

Example 12. Consider the MDs in Example 9 and the query $\mathcal{Q}(x, z) : \exists y R(x, y, z)$. The resolved answer problem for this set of MDs and query was shown to be intractable in [2, theorem 3]. The intractability arises from the fact that different minimally resolved instances can have different sets of value positions merged in the C column depending on the choice of update values in the B column. This is because updates to the B attribute values can generate *accidental similarities* between these values, i.e. similarities that result from a particular choice of update value. Depending on which value positions become similar in the B column as a result of the update, different sets of values in the C column will be merged in the subsequent update.

However, the MP-resolved answer problem is tractable in this case, because the additional restriction on updates implies that all minimally MP-resolved instances must have the same sets of value positions merged. To see this, consider the minimally MP-resolved instances of instance D of relation R, in which distinct values are dissimilar.

$R(D)$	A	B	C
t_1	a	a	a
t_2	a	c	b
t_3	b	a	c
t_4	b	d	d
t_5	c	e	e
t_6	c	f	f

Updating D according to Definition 6 merges the pairs of values at positions $\{t_1[B], t_2[B]\}$, $\{t_3[B], t_4[B]\}$, and $\{t_5[B], t_6[B]\}$. The minimal change constraint implies that the update value for each pair of positions must be the value of one of the positions. Thus, accidental similarities cannot arise between $t_5[B]$ or $t_6[B]$ and other positions in the B column.

$t_1[B]$, $t_2[B]$, $t_3[B]$, and $t_4[B]$ become accidentally similar if a is chosen as the update value for both of the pairs $\{t_1[B], t_2[B]\}$ and $\{t_3[B], t_4[B]\}$. However, this accidental similarity cannot affect subsequent updates to the values in the C column, as it could under Definition 4. This is because the first update merges positions $t_1[C]$ and $t_3[C]$. To avoid unmerging these positions in the next update, all of the positions $t_1[C]$, $t_2[C]$, $t_3[C]$, and $t_4[C]$ must be updated to a common value. Thus, the set of sets of positions to be merged is the same in all MP-resolved instances, namely $\{\{t_1[B], t_2[B]\}, \{t_3[B], t_4[B]\}, \{t_5[B], t_6[B]\}, \{t_1[C], t_2[C], t_3[C], t_4[C]\}, \{t_5[C], t_6[C]\}\}$. □

Example 12 illustrates a more general result concerning the form of MP-resolved instances for a certain class of sets of MDs. To state the result, we need the following definition.

Definition 8. [3] Consider an instance D, and set of MDs $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$. (a) For MD $m:\ R[\bar{A}] \approx R[\bar{A}] \rightarrow R[\bar{B}] \doteq R[\bar{B}]$, T_m is the reflexive, symmetric, transitive closure of the binary relation that relates pairs of tuples t_1 and t_2 in D satisfying $t_1[\bar{A}] \approx t_2[\bar{A}]$. (b) For a subset $\mathcal{M}'$ of $\mathcal{M}$, $T_{\mathcal{M}'}$ is the reflexive, symmetric, transitive closure of $\{T_m \mid m \in \mathcal{M}'\}$. □

Proposition 3. Consider a set of MDs of the form

$$
\begin{aligned}
&m_1:\ R[A_1] \approx R[A_1] \rightarrow R[A_2] \doteq R[A_2]\\
&m_2:\ R[A_2] \approx R[A_2] \rightarrow R[A_3] \doteq R[A_3]\\
&\vdots\\
&m_{n-1}:\ R[A_{n-1}] \approx R[A_{n-1}] \rightarrow R[A_n] \doteq R[A_n]
\end{aligned}
$$

Take $\mathcal{M}_{<i} := \{m_j \mid j < i\}$. For any instance D of R, the MP-resolved instances are obtained by updating, for attribute A_i, for each equivalence class E of $T_{\mathcal{M}_{<i}}$, all values of positions in the set $\{t[A_i] \ :\ t \in E\}$ to a most frequently occurring value of the positions in the set. □

Example 13. (Example 12 continued) The equivalence classes of T_{m_1} are $\{t_1, t_2\}$, $\{t_3, t_4\}$, $\{t_5, t_6\}$. By Proposition 3, the sets of merged value positions in a MP-resolved instance include $\{t_1[B], t_2[B]\}$, $\{t_3[B], t_4[B]\}$, and $\{t_5[B], t_6[B]\}$. The equivalence classes of $T_{\{m_1,m_2\}}$ are $\{t_1, t_2, t_3, t_4\}$ and $\{t_5, t_6\}$. Thus the remaining sets of values to be merged are $\{t_1[C], t_2[C], t_3[C], t_4[C]\}$ and $\{t_5[C], t_6[C]\}$, as expected. □

For certain sets of MDs, called *hit-simple-cyclic (HSC)* sets, a similar characterization to Proposition 3 has been provided for minimally resolved instances ([3], Proposition 1). In [4], this was used to derive a query rewriting method for efficiently retrieving the resolved answers to a class of queries called *unchangeable attribute join conjunctive queries (UJCQs)*. By an exactly analogous argument, UJCQs can be rewritten to efficiently retrieve the MP-resolved answers to these queries for the sets of MDs to which Proposition 3 applies.

In [3, 4], we have Datalog rewriting for resolved query answering (for certain classes of MDs and queries) for resolved query answering. A natural question is as to whether we still have that for clean query answering. And the same for clean query answering under all the clean instances (not only minimally). Actually, the answer is positive. The rewriting can be used for clean query answering for the same sets of MDs and queries. In fact, it can be extended to sets of MDs for which it did not apply in the case of resolved query answering (i.e. HSC sets with the requirement of cyclicity of the MD graph removed). For clean query answering under all clean instances, the tractable cases should be similar to those for query answering under all resolved instances.

5 Chase with Single-MD Steps

Another possible modification to the semantics of [2] is to apply the MDs one at a time to a single pair of tuples at each step rather than all MDs simultaneously. The following example shows that it can lead to infinite chase sequences even for non-interacting sets of MDs.

Example 14. Consider the relation $R[A, B]$, MD $R[A] = R[A] \rightarrow R[B] \doteq R[B]$, and instance

A	B
a	b
a	c
a	e

This relation and MD has the following infinite chase:

$$\begin{array}{|c|c|}\hline A & B \\ \hline a & b \\ a & c \\ a & e \\ \hline \end{array} \mapsto \begin{array}{|c|c|}\hline A & B \\ \hline a & b \\ a & b \\ a & e \\ \hline \end{array} \mapsto \begin{array}{|c|c|}\hline A & B \\ \hline a & b \\ a & e \\ a & e \\ \hline \end{array} \mapsto \begin{array}{|c|c|}\hline A & B \\ \hline a & b \\ a & b \\ a & e \\ \hline \end{array} \mapsto \begin{array}{|c|c|}\hline A & B \\ \hline a & b \\ a & e \\ a & e \\ \hline \end{array} \mapsto \cdots$$

□

Infinite chase sequences can be avoided as in the previous section by requiring that the chase preserve the equalities that it generates. We refer to the instances thus generated as *incremetal clean instances*. We now define what it means for a set of MDs to be satisfied under this semantics.

Definition 9. Let D, D' be instances for $\mathcal{S}$ with the same tuple ids, $m : \ R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \doteq S[\bar{E}]$ an MD and t_R and t_S tuples in $R(D)$ and $S(D)$, respectively. (D, D', t_R, t_S) satisfies m, denoted $(D, D', t_R, t_S) \vDash m$, iff:
(a) $t_R[\bar{A}] \approx t_S[\bar{B}]$ in D, and there exists a corresponding pair (C, E) of $(\bar{C}, \bar{E})$ such that $t_R[C] \neq t_S[E]$ in D and $t_R[C] = t_S[E]$ in D'.
(b) No values change in going from D to D' other than $t_R[C]$ and $t_S[E]$. □

The set of incremental instances for a given instance and set of MDs is in general different from that obtained under the semantics of [2]. More precisely, for a given instance, the set of resolved instances does not necessarily contain and is not necessarily contained in the set of incremental clean instances. This is demonstrated in the following two examples.

Example 15. This example shows that for some instances, there are resolved instances that are not incremental clean instances. The following chase produces a resolved instance D'' of the initial instance D for the given MDs.

$$R[A] = R[A] \rightarrow R[B] \doteq R[B] \tag{2}$$
$$R[B] = R[B] \rightarrow R[C] \doteq R[C] \tag{3}$$

$R(D)$	A	B	C
t_1	a	a	a
t_2	b	a	b
t_3	b	c	c
t_4	a	c	d
t_5	e	c	e

$\rightarrow$

$R(D')$	A	B	C
t_1	a	b	a
t_2	b	a	a
t_3	b	a	c
t_4	a	b	c
t_5	e	c	c

$\rightarrow$

$R(D'')$	A	B	C
t_1	a	b	a
t_2	b	a	b
t_3	b	a	b
t_4	a	b	a
t_5	e	c	c

It is easy to see that D'' is not an incremental clean instance. Since the value of $t_5[C]$ in D'' is different from the value of $t_5[C]$ in D, it must have been merged with another value in the C column. Since it is not equal to any of the other values in the C column, it must have been subsequently unmerged. □

Example 16. This example shows that for some instances, there are incremental clean instances that are not resolved instances. The following chase produces an incremental clean instance D'' of the initial instance D for the given MDs (the values b and b' are similar, and all other pairs of distinct values are dissimilar).

$$R[A] \approx R[A] \rightarrow R[B] \doteq R[B] \tag{4}$$

$$R[B] \approx R[B] \rightarrow R[C] \doteq R[C] \tag{5}$$

$R(D)$	A	B	C
t_1	a	b	a
t_2	a	b'	b
t_3	b	b	c

$\rightarrow$

$R(D')$	A	B	C
t_1	a	b	a
t_2	a	b'	a
t_3	b	b	c

$\rightarrow$

$R(D'')$	A	B	C
t_1	a	a	a
t_2	a	a	a
t_3	b	b	c

In this chase, we applied MD (4) and then MD (3) to t_1 and t_2 to obtain the stable instance D''. It is easy to see that D'' cannot be a resolved instance. This is because the first update would make all values in the C column equal, and these values could not change subsequently because they would not be modifiable. □

References

[1] J. Gardezi, L. Bertossi, and I. Kiringa. Matching dependencies with arbitrary attribute values: semantics, query answering and integrity constraints. *Proc. Int. WS on Logic in Databases (LID'11)*, ACM Press, 2011, pp. 23-30.

[2] J. Gardezi, L. Bertossi, and I. Kiringa. Matching dependencies: semantics, query answering and integrity constraints. *Frontiers of Computer Science*, Springer, 2012, 6(3):278-292.

[3] Gardezi, J. and Bertossi, L. Tractable Cases of Clean Query Answering under Entity Resolution via Matching Dependencies. Proc. International Conference on Scalable Uncertainty Management (SUM'12), Springer LNAI 7520, pp. 180.193, 2012.

[4] Gardezi, J. and Bertossi, L. Query Rewriting using Datalog for Duplicate Resolution. Proc. "The 2nd Workshop on the Resurgence of Datalog in Academia and Industry", (Datalog 2.0, 2012), Springer LNCS 7494, pp. 86.98, 2012.