The VLDB Journal (2011) 20:495–520
DOI 10.1007/s00778-010-0206-6

REGULAR PAPER

Dynamic constraints for record matching

Wenfei Fan · Hong Gao · Xibei Jia · Jianzhong Li · Shuai Ma

Received: 9 January 2010 / Revised: 21 August 2010 / Accepted: 7 October 2010 / Published online: 16 November 2010

Abstract This paper investigates constraints for matching records from *unreliable* data sources. (a) We introduce a class of *matching dependencies* (MDs) for specifying the semantics of unreliable data. As opposed to static constraints for schema design, MDs are developed for record matching, and are defined in terms of *similarity predicates* and a *dynamic semantics*. (b) We identify a special case of MDs, referred to as *relative candidate keys* (RCKs), to determine what attributes to compare and how to compare them when matching records across possibly different relations. (c) We propose a mechanism for inferring MDs, a departure from traditional implication analysis, such that when we cannot match records by comparing attributes that contain errors, we may still find matches by using other, more reliable attributes. Moreover, we develop a sound and complete system for inferring MDs. (d) We provide a quadratic-time algorithm for inferring MDs and an effective algorithm for deducing a set of high-quality RCKs from MDs. (e) We experimentally verify that the algorithms help matching tools efficiently identify keys at compile time for matching, blocking or windowing and in addition, that the MD-based techniques effectively improve the quality and efficiency of various record matching methods.

W. Fan
University of Edinburgh, Edinburgh, UK
e-mail: wenfei@inf.ed.ac.uk

X. Jia · S. Ma
School of Informatics, University of Edinburgh, Edinburgh, UK
e-mail: xibei.jia@ed.ac.uk

S. Ma
e-mail: shuai.ma@ed.ac.uk

W. Fan · H. Gao · J. Li (✉)
Harbin Institute of Technology, Harbin, Heilongjiang, China
e-mail: lijzh@hit.edu.cn

H. Gao
e-mail: honggao@hit.edu.cn

Keywords Deduction · Inference · Matching dependencies · Record matching · Data cleaning

1 Introduction

Record matching is the problem of identifying tuples in one or more relations that refer to the same real-world entity. This problem is also known as record linkage, merge-purge, data deduplication, duplicate detection and object identification. The need for record matching is evident. In data integration, it is necessary to collate information about an object from multiple data sources [31]. In data cleaning it is critical to eliminate duplicate records [7]. In master data management, one often needs to identify links between input tuples and master data [32]. The need is also highlighted by payment card fraud, which cost $4.84 billion worldwide in 2006 [27]. In fraud detection, it is a routine process to cross-check whether a card user is the legitimate card holder.

Record matching is a longstanding issue that has been studied for decades. A variety of approaches have been proposed for record matching: probabilistic (e.g., [20,29,46,48]), learning-based [13,36,42], distance-based [22], and rule-based [2,24,31] (see [15] for a recent survey).

No matter what approach to use, one often needs to decide what attributes to compare and how to compare them. Real life data is typically dirty (e.g., a person's name may appear as "Mark Clifford" and "Marx Clifford") and may not have a uniform representation for the same object in different data sources. To cope with these, it is often necessary to hinge on the semantics of the data. Indeed, domain knowledge about the data may tell us what attributes to compare. Moreover, by

(a)

	SSN	c#	FN	LN	addr	tel	email	gender	type
t_1:	111	079172485	Mark	Clifford	10 Oak Street, MH, NJ 07974	908-1111111	mc@gm.com	M	master
t_2:	222	191843658	David	Smith	620 Elm Street, MH, NJ 07976	908-2222222	dsmith@hm.com	M	visa

(b)

	c#	FN	LN	post	phn	email	gender	item	price
t_3:	111	Marx	Clifford	10 Oak Street, MH, NJ 07974	908	mc	null	iPod	169.99
t_4:	111	Marx	Clifford	NJ	908-1111111	mc	null	book	19.99
t_5:	111	M.	Clivord	10 Oak Street, MH, NJ 07974	1111111	mc@gm.com	null	PSP	269.99
t_6:	111	M.	Clivord	NJ	908-1111111	mc@gm.com	null	CD	14.99

Fig. 1 Example credit and billing relations. **a** Example credit relation I_c; **b** Example billing relation I_b

analyzing the semantics of the data, we can deduce alternative attributes to inspect such that when matching cannot be done by comparing attributes that contain errors, we may still find matches by using other, more reliable attributes. This is illustrated by the following example.

Example 1 Consider two data sources specified by the following relation schemas:

credit (c#, SSN, FN, LN, addr, tel, email, gender, type),
billing (c#, FN, LN, post, phn, email, gender, item, price).

Here, a credit tuple specifies a credit card (with number c# and type) issued to a card holder who is identified by SSN, FN (first name), LN (last name), addr (address), tel (phone), email, and gender. A billing tuple indicates that the price of a purchased item is paid by a credit card of number c#, used by a person specified in terms of name (FN, LN), gender, postal address (post), phone (phn), and email. An example instance (I_c, I_b) of (credit, billing) is shown in Fig. 1.

For payment fraud detection, one needs to check whether for any tuple t in I_c and any tuple t' in I_b, if t[c#] = t'[c#], then $t[Y_c]$ and $t'[Y_b]$ refer to the same person, where Y_c and Y_b are two attribute lists:

Y_c = [FN, LN, addr, tel, gender],
Y_b = [FN, LN, post, phn, gender].

That is, we have to determine whether the card holder (identified by $t[Y_c]$) and the card user ($t'[Y_b]$) are the same person. If t[c#] = t'[c#] but $t[Y_c]$ and $t'[Y_b]$ do not match, then the chances are that a payment card fraud has been committed.

However, due to errors in the data sources, one may not be able to match $t[Y_c]$ and $t'[Y_b]$ via pairwise comparison of their attributes, i.e., it is not straightforward to decide $t[Y_c]$ and $t'[Y_b]$ actually refer to the same person. In the instance of Fig. 1, for example, billing tuples $t_3 - t_6$ and credit tuple t_1 actually refer to the same card holder. However, *no match* can be found when we check whether the Y_b attributes of $t_3 - t_6$ and the Y_c attributes of t_1 are identical.

Domain knowledge about the data suggests that we only need to compare LN, FN and address when matching $t[Y_c]$ and $t'[Y_b]$ [24]: if a credit tuple t and a billing tuple t' have the same address and last name, and if their first names are similar (although they may not be identical), then the two tuples refer to the same person. That is, LN, FN and address, together with two equality operators and a similarity predicate $\approx_d$, are a "key" for matching $t[Y_c]$ and $t'[Y_b]$:

φ_1: If t[LN, addr] = t'[LN, post] and if t[FN] and t'[FN] are *similar w.r.t.* $\approx_d$, then $t[Y_c]$ and $t'[Y_b]$ are a match.

Such a *matching key* tells us what attributes to compare and how to compare them in order to match $t[Y_c]$ and $t'[Y_b]$. By comparing only the attributes in φ_1, we can now match t_1 and t_3, although their FN, tel, email and gender attributes are not pairwise identical.

A closer examination of the data semantics further suggests the following: for any credit tuple t and billing tuple t',

φ_2: if t[tel] = t'[phn], then we can identify t[addr] and t'[post], i.e., they should be changed by taking the same value in any uniform representation of the address.

φ_3: if t[email] = t'[email], then we can identify t[LN, FN] and t'[LN, FN].

None of these makes a key for matching $t[Y_c]$ and $t'[Y_b]$, i.e., we cannot match entire $t[Y_c]$ and $t'[Y_b]$ by just comparing their email or phone attributes. Nevertheless, putting them together with the matching key φ_1 given above, we can infer the following new matching keys:

rck$_2$: LN, FN and phone, to be compared with $=, \approx_d, =$ operators, respectively,
rck$_3$: address and email, to be compared via $=$, and
rck$_4$: phone and email, to be compared via $=$.

These three *deduced keys* have added value. While we cannot match t_1 and $t_4 - t_6$ by using the initial matching key φ_1, we can match these tuples based on the deduced keys. Indeed, using key rck$_4$, we can now match t_1 and t_6 in Fig. 1: they have the same phone and email, and can thus be identified, although their name, gender and address attributes are *radically different*. That is, although there are errors in those attributes, we are still able to match the records by inspecting their email and phone attributes. Similarly, we can match t_1 and t_4, and t_1 and t_5 using rck_2 and rck_3, respectively. □

The example highlights the need for effective techniques to specify and reason about the semantics of data in unreliable relations for record matching. One can draw an analogy of this to our familiar notion of functional dependencies (FDs). Indeed, to identify a tuple in a relation we use candidate keys. To find the keys we first specify a set of FDs, and then infer keys by the *implication analysis* of the FDs. For all the reasons that we need FDs and their reasoning techniques for identifying tuples in a clean relation, it is also important to develop (a) dependencies to specify the semantics of data in relations that may contain errors and (b) effective techniques to reason about these dependencies.

One might be tempted to use FDs in record matching. Unfortunately, FDs and other traditional dependencies are defined on clean (error-free) data, mostly for schema design (see, e.g., [1]). In contrast, for record matching, we have to accommodate errors and different representations in different data sources. As will be seen shortly, in this context, we need a form of dependencies quite *different* from their traditional counterparts, and a reasoning mechanism more *intriguing* than the standard notion of implication analysis.

The need for identifying what attributes to compare [29,45] and moreover, the need for dependencies in record matching [5,12,24,38,45] have long been recognized. It is known that matching keys typically assure *high match accuracy* [15]. However, no previous work has studied how to specify and reason about dependencies for matching records across unreliable data sources.

Contributions. This paper proposes a class of dependencies for record matching and provides techniques for reasoning about such dependencies.

(1) Our first contribution is a class of *matching dependencies* (MDs) of the form: if some attributes match then *identify* some other attributes. For instance, all the semantic relations (φ_1, φ_2, φ_3, rck_2, rck_3, rck_4) we have seen in Example 1 can be expressed as MDs. In contrast to traditional dependencies, matching dependencies have a *dynamic (update) semantics* to accommodate errors in unreliable data sources. They are defined in terms of *similarity operators* and across possibly *different relations*.

(2) Our second contribution is a formalization of matching keys, referred to as *relative candidate keys* (RCKs). RCKs are a special class of MDs that match tuples by comparing a *minimum number* of attributes. For instance, the matching keys φ_1, rck_2, rck_3 and rck_4 given in Example 1 are RCKs relative to (Y_c, Y_b). The notion of RCKs substantially differs from traditional candidate keys for relations: they aim to identify tuples across possibly different, unreliable data sources.

(3) Our third contribution is a generic reasoning mechanism for deducing MDs from a set of given MDs. For instance, the keys rck_2, rck_3 and rck_4 of Example 1 can be deduced from the MDs φ_1, φ_2 and φ_3. In light of the dynamic semantics of MDs, the reasoning is a departure from our familiar terrain of traditional dependency implication.

(4) Our fourth contribution is a sound and complete inference system for deducing MDs from a set of given MDs, along the same lines as the Armstrong's Axioms for the implication analysis of FDs (see, e.g., [1]). The inference of MDs is, however, more involved than its FDs counterpart: it consists of nine rules, instead of three axioms.

(5) Our fifth contribution is an algorithm for determining whether an MD can be deduced from a set of MDs. Despite the dynamic semantics of MDs and the use of similarity operators, the deduction algorithm is in $O(n^2)$ time, where n is the length of MDs (see e.g., [1] for discussions about the length of dependencies). This complexity bound is comparable to the traditional implication analysis of FDs.

(6) Our sixth contribution is an algorithm for deducing a set of RCKs from MDs. Recall that it takes exponential time to enumerate all candidate keys from a set of FDs [33]. For the same reason, it is unrealistic to compute all RCKs from MDs. To cope with this, we introduce a quality model such that for any given number k, the algorithm returns k high-quality RCKs *w.r.t.* the model, in $O(kn^3)$ time, where n is as above.

We remark that the reasoning is efficient: it is done at the schema level and at compile time, and n is the size of MDs (analogous to the size of FDs), which is typically much smaller than that of data on which matching is conducted.

(7) Our final contribution is an experimental study. We first evaluate the scalability of our reasoning algorithms and find them quite efficient. For instance, it takes less than 50 seconds to deduce 50 high-quality RCKs from a set of 2000 MDs. Moreover, we evaluate the impacts of RCKs on the quality and performance of two record matching methods: statistical and rule-based. Using real-life data scraped from the Web, we find that RCKs improve match quality by up to 20%, in terms of *precision* (the ratio of *true* matches correctly found to all matches returned, true or false) and *recall* (the ratio of *true* matches correctly found to all matches in the data, correctly found or incorrectly missed). In many cases, RCKs improve the efficiency as well. In addition, RCKs are also useful in blocking and windowing,

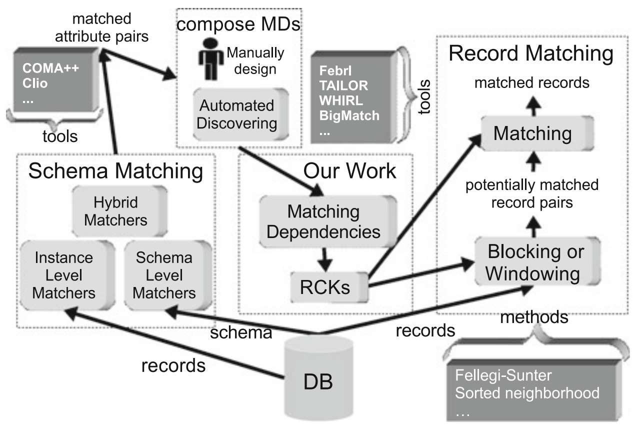

Fig. 2 Our work in record matching and schema matching context

two of the widely used optimization techniques for matching records in large relations (see below). We find that blocking and windowing based on (parts of, i.e., a subset of attributes in) RCKs consistently lead to better match quality than their counterparts without using RCKs, with 10% improvement.

Applications. As illustrated in Fig. 2, this work does not aim to introduce another record matching algorithm. It is to *complement* existing methods and to improve their match quality and efficiency, in particular when dealing with large, unreliable data sources. Taken together with automated methods for schema matching and techniques for dependency discovery, this work aims to provide effective techniques to find keys for matching, blocking and windowing.

Matching. Naturally RCKs provide matching keys: they tell us what attributes to compare and how to compare them. As observed in [29], to match tuples of arity n, there are 2^n possible comparison configurations. Thus it is unrealistic to enumerate all matching keys exhaustively and then manually select "the best keys" among possibly exponentially many candidates. In contrast, RCKs are automatically deduced from MDs *at the schema level and at compile time*. In addition, RCKs reduce the cost of inspecting a single pair of tuples by minimizing the number of attributes to compare.

Better still, RCKs improve match quality. Indeed, deduced RCKs *add value*: as we have seen in Example 1, while tuples $t_4 - t_6$ and t_1 cannot be matched by the given key, they are identified by the deduced RCKs. The added value of deduced rules has long been recognized in census data cleaning: deriving implicit rules from explicit ones is a routine practice of US Census Bureau [19,47].

Blocking. To handle large relations, it is common to partition the relations into blocks based on blocking keys (discriminating attributes), such that only tuples in the same block are compared (see, e.g., [15]). This process is often repeated multiple times to improve match quality, each using a different blocking key. The match quality is highly dependent on *the choice of blocking keys*. As shown by our experimental results, blocking can be effectively done by grouping similar tuples by (parts of) RCKs.

Windowing. An alternative way to cope with large relations is by first sorting tuples using a key, and then comparing the tuples using a sliding window of a fixed size, such that only tuples within the same window are compared [24]. As verified by our experimental study, (parts of) RCKs suffice to serve as quality sorting keys.

We contend that the MD-based techniques can be readily incorporated into matching tools to improve their quality and efficiency. Provided a small initial set of MDs, matching tools can employ the reasoning techniques to automatically derive high-quality RCKs, and use them as different kinds of keys for matching, blocking and windowing.

In this work, we focus on the specification of MDs and inference of MDs from an initial set of MDs. As shown in Fig. 2, the initial MDs may be either manually designed based on domain knowledge, or automatically discovered from sample data via a combination of schema matching techniques (e.g., the methods supported by COMA^{++} [6] or Clio [23]; see [35] for a survey), expectation maximization (EM) algorithms [29,46] and methods for dependency discovery (e.g., [17,26,40]). We defer the study of MDs discovery to future work.

We consider MDs defined on two distinct relations. In practice one often needs to identify tuples from different data sources, rather than in the same relation. The need for this is evident in data cleaning, when we need to match input data and master data, which typically have radically different schemas (see e.g., [2,37]). As will be seen in Section 4, there are subtle differences between the inference of MDs across different relations and its counterpart in a single relation. Nevertheless, most results of this work can be readily extended to MDs defined on the same relation.

Organization. The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 defines MDs and RCKs. Section 4 introduces reasoning mechanism and an inference system for MDs. Algorithms for deducing MDs and RCKs are provided in Sects. 5 and 6, respectively. The experimental study is presented in Section 7, followed by topics for future work in Sect. 8.

2 Related work

This work is an extension of [18] by including a sound and complete inference system for MDs (Sect. 4.2), and details of the algorithms and proofs in Sects. 5 and 6. The idea of this work was also presented in an invited tutorial [16], without revealing technical details.

A variety of methods (e.g., [2,11,13,19–22,24,29,31,36,42,46,47]) and tools (e.g., Febrl, TAILOR, WHIRL, BigMatch) have been developed for record matching (see [15] for a recent survey). There has also been a host of work on more general data cleaning and ETL tools (see [7] for a survey). This work is not to provide another record matching algorithm. Instead, it *complements* prior matching methods by providing dependency-based reasoning techniques to help decide keys for *matching, blocking* or *windowing*. An automated reasoning facility will effectively reduce manual effort and improve match quality and efficiency. While such a facility should logically become part of the record matching process, we are not aware of analogous functionality currently in any systems or tools.

Rules for matching are studied in [2,4,5,12,24,31,38,39,45]. A class of rules is introduced in [24], which can be expressed as relative candidate keys of this paper; in particular, the key used in Example 1 is borrowed from [24]. Extensions of [24] are proposed in [2,4], by supporting dimensional hierarchies and constant transformations to identify domain-specific abbreviations and conventions (e.g., "United States" to "USA"). It is shown that matching rules and keys plays an important role in industry-scale credit checking [45]. The need for dependencies for record matching is also highlighted in [12,38]. A class of *constant* keys is studied in [31], to match records in a single relation. Recursive algorithms are developed in [5,39], to compute matches based on certain dependencies. The AJAX system [21] also advocates matching transformations specified in a declarative language. However, to the best of our knowledge, no previous work has formalized matching rules or matching keys as dependencies in a logic framework, or has studied automated techniques and inference systems for reasoning about dependencies for record matching. This work provides the first formal specifications and static analyses of matching rules, to deduce keys for matching, blocking and windowing via automated reasoning of dependencies.

There are other approaches [29,44,46] to deciding what attributes to compare in object identification. A heuristic method was proposed in [44] to identify relevant elements in XML documents, by capturing their structural similarity such as navigational paths. It differs from this work in that it does not consider automated techniques for deducing matching rules. Probabilistic methods have also been studied in [29,46], using an expectation maximization (EM) algorithm. In contrast, this work decides what attributes to compare by the static analyses of MDs at the schema level and at compile time. As will be seen in Section 7, the MD-based method outperforms the EM-based approach in both accuracy and efficiency. On the other hand, the two approaches *complement* each other: one can first *discover* a small set of MDs via sampling and learning, and then leverage the reasoning techniques to deduce RCKs. It should be remarked to get an initial set of MDs one can also leverage *domain knowledge analysis*, along the same lines as the design of FDs.

As remarked earlier, it is important to develop techniques for discovering matching dependencies. A variety of methods have been proposed for discovering FDs (e.g., [17,26]) and traditional keys (e.g., [40]), which are defined on a single relation in terms of equality. However, to discover MDs these methods need substantial extensions, to decide what attributes across different schemas correspond to each other and what similarity operators should be used to compare them. As suggested in Fig. 2, discovery algorithms for MDs need to leverage schema matching techniques [6,35] and expectation maximization algorithms [29,46]. Discovery methods for MDs are not the focus of this work.

Dependency theory is almost as old as the study of relational databases itself. Traditional dependencies, e.g., FDs, are first-order logic sentences in which domain-specific similarity metrics are *not* expressible. Furthermore, these dependencies are static constraints for which updates are not a concern and are studied for schema design on clean data (see, e.g., [1] for a detailed discussion of relational dependencies). In contrast, for matching records from unreliable data sources, one needs similarity metrics to accommodate errors in the data. In addition, as will be seen shortly, the static semantics of traditional dependencies is no longer appropriate in record matching. Indeed, the semantics of MDs and the notion of their deductions are a departure from their traditional counterparts for dependencies and implication.

There have been extensions of FDs by supporting similarity predicates [9,30]. There has also be been work on schema design for uncertain relations by extending FDs [3]. Like traditional FDs, these extensions are defined on a single relation and have a static semantics. They are quite different from MDs studied in this work, which are defined across possibly different relations and have a dynamic semantics.

Dynamic constraints have been studied for database evolution [43] and for XML updates [10]. These constraints aim to express an invariant connection between the old value and the new value of a data element when the data is updated. They differ from MDs in that these constraints are restrictions on how given updates should be carried out. In contrast, MDs specify how data elements should be identified for record matching. In other words, MDs are to determine what (implicit) updates are necessary for identifying records. Furthermore, similarity predicates are not supported by the constraints of [10,43].

3 Matching dependencies and relative candidate keys

In this section, we first define matching dependencies and then present the notion of relative candidate keys. For the

Table 1 Symbols and abbreviations

Θ	A fixed set of similarity predicates, including '='
$\approx$	A similarity predicate in Θ
$\approx_d$	A predicate in Θ defined in terms of edit distance
$\rightleftharpoons$	The matching operator
MD	Matching dependency
RCK	Relative candidate key
LHS	The left-hand side
RHS	The right-hand side
I	A relation instance
D	A database instance
$I \sqsubseteq I'$	I and I' are two instances of the same relation schema, and for each tuple t in I there is a tuple t' in I' such that t and t' have the same tuple id
$(I_1, I_2) \sqsubseteq (I'_1, I'_2)$	$I_1 \sqsubseteq I'_1$ and $I_2 \sqsubseteq I'_2$
X, Y, Z, U, V, W	Each one denotes a list of attributes, respectively
A, B, C, E, F, G, H	Each one denotes a single attribute, respectively
φ, ϕ, ψ or γ	Each one denotes a MD or RCK, respectively
Σ or Γ	Denote a set of MDs or RCKs

readers' convenience, some symbols and abbreviations to be used are listed in Table 1.

3.1 Matching dependencies

Let R_1 and R_2 be two relation schemas, and Y_{R_1} and Y_{R_2} be two lists of attributes in R_1 and R_2, respectively. The record matching problem is stated as follows.

Given an instance (I_1, I_2) of (R_1, R_2), *the record matching problem* is to identify all tuples $t_1 \in I_1$ and $t_2 \in I_2$ such that $t_1[Y_{R_1}]$ and $t_2[Y_{R_2}]$ refer to the same real-world entity.

Observe the following. (a) Even when $t_1[Y_{R_1}]$ and $t_2[Y_{R_2}]$ refer to the same entity, one may still find that $t_1[Y_{R_1}] \neq t_2[Y_{R_2}]$ due to errors or different representations in the data. (b) The problem aims to match $t_1[Y_{R_1}]$ and $t_2[Y_{R_2}]$, i.e., parts of t_1 and t_2 specified by lists of attributes, not necessarily the entire tuples t_1 and t_2. (c) It is to find matches across relations of possibly different schemas.

To accommodate these in record matching, we define MDs in terms of similarity predicates, a departure from our familiar FDs. Before we define MDs, we first discuss similarity predicates. To simplify the discussion, we assume that R_1 and R_2 specify distinct data sources. Nevertheless, the results of this paper can be readily adapted to the context where R_1 and R_2 denote the same relation.

Similarity predicates. Assume a fixed set Θ of domain-specific similarity relations. For each $\approx$ in Θ, and values x, y in the specific domains in which $\approx$ is defined, we write $x \approx y$ if (x, y) is in $\approx$, and refer to $\approx$ as a *similarity predicate*. The predicate can be defined in terms of any similarity metric used in record matching, e.g., q-grams, Jaro distance or edit distance (see [15] for a survey), such that $x \approx y$ is true if x and y are "close" enough *w.r.t.* a predefined threshold.

In particular, the equality relation $=$ is in Θ.

We also use a *matching operator* $\rightleftharpoons$: for any values x and y, $x \rightleftharpoons y$ indicates that x and y are identified via updates, i.e., we update x and y to make them identical. The semantics of the operator $\rightleftharpoons$ will be elaborated shortly.

In terms of similarity predicates and the matching operator, we next define matching dependencies.

Matching dependencies. A *matching dependency* (MD) φ for (R_1, R_2) is syntactically defined as follows:

$$\bigwedge_{j\in[1,k]} (R_1[A_j] \approx_j R_2[B_j]) \rightarrow \bigwedge_{i\in[1,h]} (R_1[E_i] \rightleftharpoons R_2[F_i]),$$

where (1) for each $j \in [1, k]$, A_j and B_j are attributes of R_1 and R_2, respectively, with the same domain; similarly for E_i and F_i when $i \in [1, h]$; and (2) $\approx_j$ is a similarity predicate in Θ that is defined in the domain of $R_1[A_j]$ and $R_2[B_j]$.

Let Z_1 be the list $[E_1, \ldots, E_h]$, and $Z_2 = [F_1, \ldots, F_h]$. Then intuitively, the MD φ states that if for all $j \in [1, k]$, $R_1[A_j]$ and $R_2[B_j]$ are similar *w.r.t.* the similarity predicate $\approx_j$, then $R_1[Z_1]$ and $R_2[Z_2]$ refer to the same object and should be identified (made identical). We write φ as

$$\bigwedge_{j\in[1,k]} (R_1[A_j] \approx_j R_2[B_j]) \rightarrow R_1[Z_1] \rightleftharpoons R_2[Z_2],$$

We refer to $\bigwedge_{j\in[1,k]}(R_1[A_j] \approx_j R_2[B_j])$ and $R_1[Z_1] \rightleftharpoons R_2[Z_2]$ as the LHS and the RHS of φ, respectively.

Example 2 The semantic relations given in Example 1 can be expressed as MDs, as follows:

$$\varphi_1 : \text{credit}[\text{LN}] = \text{billing}[\text{LN}] \wedge \text{credit}[\text{addr}] = \text{billing}[\text{post}] \wedge \text{credit}[\text{FN}] \approx_d \text{billing}[\text{FN}] \rightarrow \text{credit}[Y_c] \rightleftharpoons \text{billing}[Y_b]$$

$$\varphi_2 : \text{credit}[\text{tel}] = \text{billing}[\text{phn}] \rightarrow \text{credit}[\text{addr}] \rightleftharpoons \text{billing}[\text{post}]$$

$$\varphi_3 : \text{credit}[\text{email}] = \text{billing}[\text{email}] \rightarrow \text{credit}[\text{FN, LN}] \rightleftharpoons \text{billing}[\text{FN, LN}]$$

where φ_1 states that for any credit tuple t and billing tuple t', if t and t' have the same last name and address, and if their first names are similar *w.r.t.* $\approx_d$ (but may not necessarily be identical), then $t[Y_c]$ and $t'[Y_b]$ should be identified. Similarly, if t and t' have the same phone number then we should identify their addresses (φ_2); and if t and t' have the same email then their names should be identified (φ_3). Note that while name, address and phone are part of Y_b and Y_c, email is *not*, i.e., the LHS attributes of an MD is neither necessarily contained in nor disjoint from its RHS attributes. □

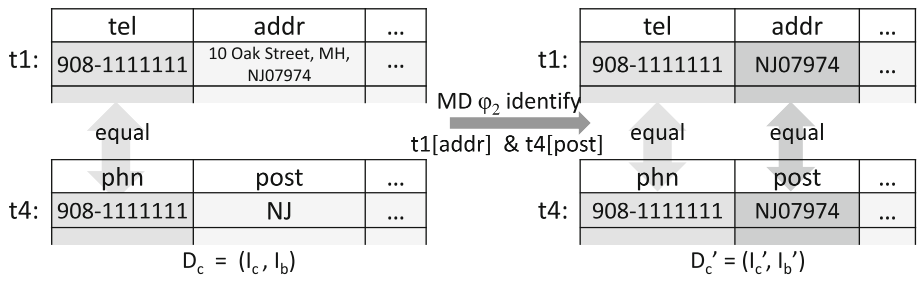

Fig. 3 MDs expressing matching rules

To simplify the discussion, we assume *w.l.o.g.* that $R_1[A_j]$ and $R_2[B_j]$ have the same domain, which can be achieved by data standardization (see [15] for details).

Dynamic semantics. Recall that a functional dependency (FD) $X \rightarrow Y$ simply assures that for any tuples t_1 and t_2, if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$. In contrast, to accommodate unreliable data, the semantics of MDs is more involved. To present the semantics we need the following notations.

Extensions. To keep track of tuples during a matching process, we assume a temporary unique tuple id for each tuple. For instances I and I' of the same schema, we write $I \sqsubseteq I'$ if for each tuple t in I there is a tuple t' in I' such that t and t' have the same tuple id. Here t' is an updated version of t, and t' and t may differ in some attribute values.

For two instances $D = (I_1, I_2)$ and $D' = (I_1', I_2')$ of (R_1, R_2), we write $D \sqsubseteq D'$ if $I_1 \sqsubseteq I_1'$ and $I_2 \sqsubseteq I_2'$.

For tuples $t_1 \in I_1$ and $t_2 \in I_2$, we write $(t_1, t_2) \in D$.

LHS matching. We say that $(t_1, t_2) \in D$ *match* the LHS of MD φ if for each $j \in [1, k]$, $t_1[A_j] \approx_j t_2[B_j]$. Intuitively, $t_1[A_j]$ and $t_2[B_j]$ pairwise satisfy the similarity predicate $\approx_j$.

For example, t_1 and t_3 of Fig. 1 match the LHS of φ_1 of Example 2: t_1 and t_3 have identical LN and address, and "Mark" $\approx_d$ "Marx" when $\approx_d$ is a similarity predicate defined in terms of the edit distance metric.

Semantics. We are now ready to give the semantics. Consider a pair (D, D') of instances of (R_1, R_2), where $D \sqsubseteq D'$.

The pair (D, D') of instances *satisfy* MD φ, denoted by $(D, D') \models \varphi$, if for any tuples $(t_1, t_2) \in D$, if (t_1, t_2) match the LHS of φ in the instance D, then *in the other instance* D', (a) $t_1[Z_1] = t_2[Z_2]$, i.e., the RHS attributes of φ in t_1 and t_2 are identified; and (b) (t_1, t_2) also match the LHS of φ.

Intuitively, the semantics states how φ is *enforced* as a *matching rule*: whenever (t_1, t_2) in an instance D match the LHS of φ, $t_1[Z_1]$ and $t_2[Z_2]$ ought to be made equal. The outcome of the enforcement is reflected in the other instance D'. That is, some value v is to be found such that $t_1[Z_1] = v$ and $t_1[Z_2] = v$ in D', although v is not explicitly specified.

Example 3 Consider the MD φ_2 of Example 2 and the instance $D_c = (I_c, I_b)$ of Fig. 1, in which (t_1, t_4) match the LHS of φ_2. As depicted in Fig. 3, the enforcement of φ_2 yields another instance $D_c' = (I_c', I_b')$ in which $t_1[\text{addr}] = t_4[\text{post}]$, while $t_1[\text{addr}]$ and $t_4[\text{post}]$ are *different* in D_c.

The $\rightleftharpoons$ operator only requires that $t_1[\text{addr}]$ and $t_4[\text{post}]$ are identified, but does not specify how they are updated. That is, in any D_c' that extends D_c, if (a) $t_1[\text{addr}] = t_4[\text{post}]$ and (t_1, t_4) match the LHS of φ_2 in D_c', and (b) similarly for $t_1[\text{addr}]$ and $t_6[\text{post}]$ are identified in D_c', then φ_2 is considered enforced on D_c', i.e., $(D_c, D_c') \models \varphi_2$. □

It should be clarified that we use updates just to give the semantics of MDs. In the matching process instance D may *not* be updated, i.e., there is *no* destructive impact on D.

Matching dependencies (MDs) are quite *different* from traditional dependencies such as FDs.

– MDs have "dynamic" semantics to accommodate errors and different representations in the data: if for all $j \in [1, k]$, attributes $t_1[A_j]$ and $t_2[B_j]$ match in instance D, then $t_1[Z_1]$ and $t_2[Z_2]$ are updated and identified. Here $t_1[Z_1]$ and $t_2[Z_2]$ are equal in *another instance* D' that results from the updates to D, although they may be *radically different* in the original instance D. In contrast, FDs have a "static" semantics: if certain attributes are equal in D, then some other attributes must be equal in *the same* instance D.
– MDs are defined with *similarity predicates* and the matching operator $\rightleftharpoons$, whereas FDs are defined with equality only.
– MDs are defined across possibly different relations, while FDs are defined on a single relation.

Example 4 Consider two FDs defined on schema $R(A, B, E)$:

$$f_1 : A \rightarrow B, \quad f_2 : B \rightarrow E.$$

Consider instances I_0 and I_1 of R shown in Fig. 4. Then s_1 and s_2 in I_0 violate $f_1 : s_1[A] = s_2[A]$ but $s_1[B] \neq s_2[B]$; similarly, s_1 and s_2 in I_1 violate f_2.

In contrast, consider two MDs defined on R:

$$\psi_1 : R[A] = R[A] \rightarrow R[B] \rightleftharpoons R[B],$$
$$\psi_2 : R[B] = R[B] \rightarrow R[E] \rightleftharpoons R[E],$$

where ψ_1 states that for any (s_1, s_2), if $s_1[A] = s_2[A]$, then $s_1[B]$ and $s_2[B]$ should be identified; similarly for ψ_2.

Let $D_0 = (I_0, I_0)$ and $D_1 = (I_1, I_1)$. Then $(D_0, D_1) \models \psi_1$. While $s_1[A] = s_2[A]$ but $s_1[B] \neq s_2[B]$ in I_0, s_1 and s_2 are

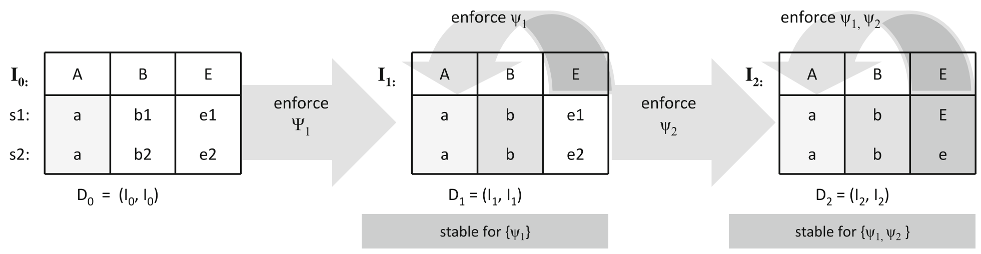

Fig. 4 The dynamic semantics of MDs

not treated a violation of ψ_1. Instead, a value b is found such that $s_1[B]$ and $s_2[B]$ are changed to b, which results in instance I_1. This is how MDs accommodate errors in unreliable data sources. Note that $(D_0, D_1) \models \psi_2$ since $s_1[B] \neq s_2[B]$ in I_0, i.e., (s_1, s_2) does not match the LHS of ψ_2 in I_0. □

A pair (D, D') of instances *satisfy* a set Σ of MDs, denoted by $(D, D') \models \Sigma$, if $(D, D') \models \varphi$ for all $\varphi \in \Sigma$.

3.2 Relative candidate keys

To decide whether $t_1[Y_{R_1}]$ and $t_2[Y_{R_2}]$ refer to the same entity, it is natural to consider a minimal number of attributes to compare. In light of this, we identify a special case of MDs.

A *key* γ *relative to* attributes (Y_{R_1}, Y_{R_2}) of (R_1, R_2) is an MD in which the RHS is fixed to be (Y_{R_1}, Y_{R_2}), i.e., an MD of the form $\bigwedge_{j\in[1,k]}(R_1[A_j] \ \approx_j \ R_2[B_j]) \rightarrow R_1[Y_{R_1}] \rightleftharpoons R_2[Y_{R_2}]$. We simply write γ as

$$((A_1, B_1, \approx_1), \ldots, (A_k, B_k, \approx_k)),$$

when R_1, R_2 and (Y_{R_1}, Y_{R_2}) are clear from the context. We refer to k as the *length* of γ.

Intuitively, γ assures that for any tuples (t_1, t_2) of (R_1, R_2), to identify $t_1[Y_{R_1}]$ and $t_2[Y_{R_2}]$ one only needs to check whether $t_1[A_j]$ and $t_2[B_j]$ satisfy the predicate $\approx_j$ for all $j \in [1, k]$. This is analogous to a relational key: to identify two tuples in a relation it suffices to inspect only their attributes in the key. Observe that there are possibly multiple keys for a relation, similarly for relative keys.

We now define RCKs. Intuitively, γ is an RCK if no other key γ' relative to (Y_{R_1}, Y_{R_2}) inspects less attributes.

The key γ is a relative *candidate* key (RCK) if there is no other key $\gamma' = ((A'_1, B'_1, \approx'_1), \ldots, (A'_l, B'_l, \approx'_l))$ relative to (Y_{R_1}, Y_{R_2}) such that (1) the length l of γ' is less than the length k of γ, and (2) for each $i \in [1, l]$, $(A'_i, B'_i, \approx'_i)$ is in γ, i.e., it is an element $(A_j, B_j, \approx_j)$ in γ for some $j \in [1, k]$.

We write $\gamma' \preceq \gamma$ if conditions (1) and (2) are satisfied.

Intuitively, to identify $t_1[Y_{R_1}]$ and $t_2[Y_{R_2}]$, an RCK specifies a *minimum* set of attributes to inspect and tells us *how* to compare these attributes via similarity predicates.

Example 5 Candidate keys relative to (Y_c, Y_b) include:

rck$_1$: ((LN, LN, =), (addr, post, =), (FN, FN, $\approx_d$))

rck$_2$: ((LN, LN, =), (tel, phn, =), (FN, FN, $\approx_d$))

rck$_3$: ((email, email, =), (addr, post, =))

rck$_4$: ((email, email, =), (tel, phn, =))

Here the key rck$_4$ states that for any credit tuple t and any billing tuple t', if t[email, tel] = t'[email, phn], then $t[Y_c]$ and $t'[Y_b]$ match; similarly for rck$_1$, rck$_2$ and rck$_3$. We also remark that email is not part of Y_b or Y_c. □

One can draw an analogy of RCKs to the familiar notion of keys for relations: both notions attempt to provide an invariant connection between tuples and the real-world entities they represent. However, there are *sharp differences* between the two notions. First, RCKs bring domain-specific *similarity* predicates into the play, carrying a *comparison vector*. Second, RCKs are defined *across different relations*; in contrast, keys are defined on a single relation. Third, RCKs have a *dynamic semantics* and aim to identify *unreliable* data, a departure from the classical dependency theory.

4 Reasoning about matching dependencies

Implication analysis of FDs can be found in almost every database textbook. Along the same lines we naturally want to deduce MDs from a set of given MDs. However, as opposed to traditional dependencies, MDs are defined in terms of domain-specific similarity predicates and matching operators, and they have dynamic semantics. As a result, traditional implication analysis no longer works for MDs.

Below we first propose a generic mechanism to deduce MDs, independent of any particular similarity predicates. We then present a sound and complete inference system for MDs, which provides algorithmic insight into the deduction of MDs. Some logical symbols used in this section are listed in Table 2.

4.1 A generic reasoning mechanism

A new challenge encountered when reasoning about MDs involves similarity predicates in MDs, which are domain-specific. In light of these, our reasoning mechanism is necessarily generic.

Table 2 Logic symbols

$(D, D') \models \varphi$	$D = (I_1, I_2) \sqsubseteq D' = (I'_1, I'_2)$, and for any tuples $t_1 \in I_1$ and $t_2 \in I_2$, if (t_1, t_2) match the LHS of φ, then *in the other instance* D', (a) t_1 and t_2 are identified on the RHS attributes of φ; and (b) (t_1, t_2) also match the LHS of φ
$(D, D') \models \Sigma$	$(D, D') \models \varphi$ for all MDs $\varphi \in \Sigma$.
$\Sigma \models_m \varphi$	For any instance D of (R_1, R_2), and *for each* stable instance D' for Σ with $(D', D') \models \Sigma$, if $(D, D') \models \Sigma$, then $(D, D') \models \varphi$.
$\mathcal{I}$	The inference system for reasoning MDs
$\Sigma \vdash_{\mathcal{I}} \varphi$	MD φ is provable from Σ using inference rules in $\mathcal{I}$
$(\Sigma, \varphi)^+$	The *closure* $(\Sigma, \varphi)^+$ of a set $\Sigma \cup \{\varphi\}$ of MDs

Generic axioms. We assume only *generic axioms* for each similarity predicate $\approx$ in Θ as follows.

– It is reflexive, i.e., $x \approx x$.
– It is symmetric, i.e., if $x \approx y$ then $y \approx x$.
– It subsumes equality, i.e., if $x = y$ then $x \approx y$.

Nevertheless, except equality =, $\approx$ is *not* assumed transitive in general, i.e., from $x \approx y$ and $y \approx z$ it does *not* necessarily follow that $x \approx z$.

The equality relation = is reflexive, symmetric and transitive, as usual. In addition, for any similarity predicate $\approx$ and values x and y, if $x \approx y$ and $y = z$, then $x \approx z$.

To simplify the discussion we also assume the following. (1) There is a unique similarity predicate $\approx_A$ defined on each distinct (infinite) domain $\mathsf{dom}(A)$. (2) The similarity predicate $\approx_A$ is *dense*: for any number k, there exist values $v, v_1, \ldots, v_k \in \mathsf{dom}(A)$ such that $v \approx_A v_i$ for $i \in [1, k]$, and $v_i \not\approx_A v_j$ for all $i, j \in [1, k]$ and $i \neq j$. That is, there are unboundedly many distinct values that are within a certain distance *w.r.t.* $\approx_A$, but are not similar to each other.

Many similarity predicates commonly found in practice are dense, e.g., edit distance. However, the linear ordering in a numeric domain may not be dense. As will be seen (in the Appendix), the proofs of Section 4.2 leverage the density to avoid subtleties introduced by similarity predicates.

The limitations of implication analysis. Another challenge is posed by the dynamic semantics of MDs. Recall the notion of implication (see, e.g., [1]): given a set Γ of traditional dependencies and another dependency ϕ, Γ *implies* ϕ if for any database D that satisfies Γ, D also satisfies ϕ. For an example of our familiar FDs, if Γ consists of $X \to Y$ and $Y \to Z$, then it implies $X \to Z$. However, this notion of implication is no longer applicable to MDs on unreliable data, as illustrated below.

Example 6 Let Σ_0 be the set $\{\psi_1, \psi_2\}$ of MDs and Γ_0 the set $\{f_1, f_2\}$ of FDs given in Example 4. Consider additional MD and FD given below:

$$\text{MD } \psi_3 : R[A] = R[A] \to R[E] \rightleftharpoons R[E],$$

$$\text{FD } f_3 : A \to E.$$

Then Γ_0 implies f_3, but Σ_0 *does not* imply ψ_3. To see this, consider I_0 (D_0) and I_1 (D_1) in Fig. 4. Observe the following.

(1) $(D_0, D_1) \models \Sigma_0$ but $(D_0, D_1) \not\models \psi_3$. Indeed, $(D_0, D_1) \models \psi_1$ and $(D_0, D_1) \models \psi_2$. However, $(D_0, D_1) \not\models \psi_3$: while $s_1[A] = s_2[A]$ in D_0, $s_1[E] \neq s_2[E]$ in D_1. This tells us that Σ_0 does *not* imply ψ_3 if the notion of implication is used for MDs.
(2) In contrast, neither I_0 nor I_1 contradicts to the implication of f_3 from Γ_0. Note that $I_0 \not\models f_3 : s_1[A] = s_2[A]$ but $s_1[E] \neq s_2[E]$. That is, s_1 and s_2 violate f_3. However, I_0 does not satisfy Γ_0 either. Indeed, $I_0 \not\models f_1 : s_1[A] = s_2[A]$ but $s_1[B] \neq s_2[B]$. Thus the conventional implication of FDs remains valid on I_0; similarly for I_1. □

Deduction. To capture the dynamic semantics of MDs in the deduction analysis, we need the following notion.

An instance D of (R_1, R_2) is said to be *stable* for a set Σ of MDs if $(D, D) \models \Sigma$. Intuitively, a stable instance D is an ultimate outcome of enforcing Σ: each and every rule in Σ is enforced until no more updates have to be conducted.

Example 7 As illustrated in Fig. 4, D_2 is a stable instance for Σ_0 of Example 6. It is an outcome of enforcing MDs in Σ_0 as matching rules: when ψ_1 is enforced on D_0, it yields another instance in which $s_1[B] = s_2[B]$, e.g., D_1. When ψ_2 is further enforced on D_1, $s_1[E]$ and $s_2[E]$ are identified, yielding D_2. Now $(D_2, D_2) \models \Sigma_0$, i.e., no further changes are necessary for enforcing the MDs in Σ_0. □

We are now ready to formalize the notion of deductions.

For a set Σ of MDs and another MD φ on (R_1, R_2), φ is said to be *deduced* from Σ, denoted by $\Sigma \models_m \varphi$, if for any instance D of (R_1, R_2), and *for each* stable instance D' for Σ, if $(D, D') \models \Sigma$ then $(D, D') \models \varphi$.

Intuitively, stable instance D' is a "fixpoint" reached by enforcing Σ on D. There are possibly many such stable instances, depending on how D is updated. The deduction analysis inspects *all* of the stable instances for Σ.

The notion of deductions is generic: no matter how MDs are interpreted, if Σ is enforced, then so must be φ. In other words, φ is a *logical consequence* of the given MDs in Σ.

Example 8 As will be seen in Section 4.2, for Σ_0 and ψ_3 given in Example 6, $\Sigma_0 \models_m \psi_3$. In particular, for the instance D_0 and the stable instance D_2 of Example 7, one can see that $(D_0, D_2) \models \Sigma_0$ and $(D_0, D_2) \models \psi_3$. □

The *deduction problem* for MDs is to determine, given any set Σ of MDs defined on (R_1, R_2) and another MD φ defined on (R_1, R_2), whether $\Sigma \models_m \varphi$.

Added value of deduced MDs. While the dynamic semantics of MDs makes it difficult to reason about MDs, it yields *added value* of deduced MDs. Indeed, while tuples in unreliable relations may not be matched by a given set Σ of MDs, they may be identified by an MD φ deduced from Σ. In contrast, when a traditional dependency ϕ is *implied* by a set of dependencies, any database that violates ϕ cannot possibly satisfy all the given dependencies.

Example 9 Let D_c be the instance of Fig. 1, and Σ_1 consist of $\varphi_1, \varphi_2, \varphi_3$ of Example 2. As shown in Example 1, (t_1, t_6) in D_c can be matched by rck_4 of Example 5, but cannot be directly identified by Σ_1. Indeed, one can easily find an instance D' such that $(D_c, D') \models \Sigma_1$ but $t_1[Y_c] \neq t_6[Y_b]$ in D'. In contrast, there is no D' such that $(D_c, D') \models \mathsf{rck}_4$ but $t_1[Y_c] \neq t_6[Y_b]$ in D'. As will be seen in Example 11, it is from Σ_1 that rck_4 is deduced. This shows that while tuples may not be matched by a set Σ of given MDs, they can be identified by MDs deduced from Σ.

The deduced rck_4 would not have had added value if the MDs were interpreted with a static semantics like FDs. Indeed, t_1 and t_6 have *radically different* names and addresses, and would be considered as a violation of rck_4 if rck_4 were treated as an "FD". At the same time, they would violate φ_1 in Σ_1. Thus with the conventional implication analysis, rck_4 would not be able to identify tuples that Σ_1 fails to match. □

4.2 A sound and complete inference system for MDs

Armstrong's Axioms have proved extremely useful in the implication analysis of FDs (see, e.g., [1]). Along the same lines one naturally wants a finite inference system that is *sound and complete* for the deduction analysis of MDs.

The inference of MDs is, however, more involved than its FDs counterpart. (1) The matching operator $\rightleftharpoons$ updates data to identify data elements. It interacts with equality $=$: $u \rightleftharpoons v$ entails that $u = v$ in the updated data. (2) Similarity predicates also interact with equality $=$, e.g., if $u = v$ then $u \approx v$, and if $u = v$ and $v \approx w$ then $u \approx w$.

Weak MDs. To show an MD is provable from a given set of MDs, we need to keep the intermediate results in the process. To capture these interactions in the deduction analysis, we introduce a weak form of MDs to express intermediate results encountered in the inference. A weak MD allows similarity predicate to appear in the RHS, in contrast to the matching operator $\rightleftharpoons$ as found in MDs. More specifically, a *weak* MD over (R_1, R_2) has one of the following forms:

$$\varphi_1 = \bigwedge_{j\in[1,k]} (R_1[A_j] \approx_j R_2[B_j]) \rightarrowtail R_1[A] \approx R_2[B],$$

$$\varphi_2 = \bigwedge_{j\in[1,k]} (R_1[A_j] \approx_j R_2[B_j]) \rightarrowtail R_i[A] \approx' R_i[B],$$

where $\approx$ and $\approx'$ are similarity predicates in Θ rather than $\rightleftharpoons$. While the RHS of φ_1 refers to two relations R_1 and R_2, the RHS of φ_2 indicates the same relation R_i for $i \in [1, 2]$ (i.e., either R_1 or R_2). We use $\rightarrowtail$ instead of $\to$ to explicitly distinguish weak MDs from MDs.

The semantics of weak MDs is a variation of its MD counterpart. Let (D, D') be a pair of instances of (R_1, R_2), where $D \sqsubseteq D'$. The pair (D, D') of instances *satisfy weak* MD φ_1, denoted by $(D, D') \models \varphi_1$, if for any tuples $(t_1, t_2) \in D$, if (t_1, t_2) match the LHS of φ_1 in D, then in D', $t_1[A] \approx t_2[B]$ and moreover, (t_1, t_2) also match the LHS of φ_1.

Similarly, $(D, D') \models \varphi_2$ if for any tuples $(t_1, t_2) \in D$, if (t_1, t_2) match the LHS(φ_2) in D, then in D', $t_i[A] \approx t_i[B]$, where t_i is either t_1 or t_2, and (t_1, t_2) also match LHS(φ_2).

To illustrate the need for weak MDs, let us consider the following example.

Example 10 Consider a set $\Sigma = \{\psi_1, \psi_2\}$ of MDs and another MD ψ, where

$$\psi_1 := R_1[A] \approx R_2[B] \to R_1[CC] \rightleftharpoons R_2[DE],$$
$$\psi_2 := R_1[A] \approx R_2[B] \wedge R_1[F] \approx R_2[D] \to R_1[G] \rightleftharpoons R_2[H], \text{ and}$$
$$\psi := R_1[A] \approx R_2[B] \wedge R_1[F] \approx R_2[E] \to R_1[G] \rightleftharpoons R_2[H].$$

The MD ψ_1 tells us that for an R_1 tuple t_1 and an R_2 tuple t_2, if $t_1[A] \approx t_2[B]$, then $t_1[CC]$ and $t_2[DE]$ should be identified; similarly, ψ_2 assures that if $t_1[AF] \approx t_2[BD]$, then $t_1[G]$ and $t_2[H]$ stand for the same entity. To show that ψ can deduced from Σ, we need the following generic reasoning.

(a) ψ_1' : $R_1[A] \approx R_2[B] \rightarrowtail R_2[D] = R_2[E]$, by (ψ_1);
(b) $\psi_2' := R_1[A] \approx R_2[B] \wedge R_1[F] \approx R_2[E] \rightarrowtail R_1[AF] \approx R_2[BE]$, by (ψ_1');
(c) ψ_3' : $R_1[A] \approx R_2[B] \wedge R_1[F] \approx R_2[E] \rightarrowtail R_1[AF] \approx R_2[BD]$, by (ψ_1', ψ_2');
(d) ψ, by (ψ_2, ψ_3').

Observe that ψ_1', ψ_2' and ψ_3' are weak MDs. Indeed, the RHS of each of them is defined in terms of *similarity operators* on attributes that are possibly in *the same relation* (e.g., ψ_1'). Hence they are not expressible as MDs of Section 3. Such weak MDs are needed in the deduction analysis. □

MD_1: If $\bigwedge_{i\in[1,\,k]}(R_2[B_i] \approx_i R_1[A_i]) \to R_2[Z_2] \rightleftharpoons R_1[Z_1]$,
then $\bigwedge_{i\in[1,\,k]}(R_1[A_i] \approx_i R_2[B_i]) \to R_1[Z_1] \rightleftharpoons R_2[Z_2]$.

MD_2: Let $L = \bigwedge_{i\in[1,\,k]}(R_1[A_i] \approx_i R_2[B_i])$. For each $i \in [1,k]$,
(1) $L \to R_1[A_i] \rightleftharpoons R_2[B_i]$ if $\approx_i$ is $=$, and
(2) $L \rightarrowtail R_1[A_i] \approx_i R_2[B_i]$ otherwise.

MD_3: If $\phi = \bigwedge_{i\in[1,\,k]}(R_1[A_i] \approx_i R_2[B_i]) \to R_1[Z_1] \rightleftharpoons R_2[Z_2]$, then for any attributes (A,B) having the same domain over (R_1, R_2) and similarity predicate $\approx$ in Θ,
(1) $\mathsf{LHS}(\phi) \bigwedge (R_1[A] \approx R_2[B]) \to \mathsf{RHS}(\phi)$, and
(2) $\mathsf{LHS}(\phi) \bigwedge (R_1[A] = R_2[B]) \to \mathsf{RHS}(\phi) \bigwedge (R_1[A] \rightleftharpoons R_2[B])$.

MD_4: Let $L = \bigwedge_{j\in[1,\,g]}(R_1[E_j] \approx_j R_2[F_j])$ such that for each $j \in [1,\,g]$, $\approx_j$ is not $=$.
If (1) $\phi_1 = \bigwedge_{i\in[1,\,k]}(R_1[A_i] \approx_i R_2[B_i]) \to R_1[W_1] \rightleftharpoons R_2[W_2]$,
(2) $\phi_2 = L \bigwedge (R_1[W_1] = R_2[W_2]) \to R_1[Z_1] \rightleftharpoons R_2[Z_2]$, and
(3) for each $j \in [1,g]$, $\mathsf{LHS}(\phi_1) \rightarrowtail R_1[E_j] \approx_j R_2[F_j]$,
then $\mathsf{LHS}(\phi_1) \bigwedge L \to \mathsf{RHS}(\phi_2)$.

MD_5: Let $L = \bigwedge_{i\in[1,\,k]}(R_1[A_i] \approx_i R_2[B_i])$.
If $\phi = L \bigwedge (R_1[A] \approx R_2[B]) \to R_1[Z_1] \rightleftharpoons R_2[Z_2]$,
then $L \bigwedge (R_1[A] = R_2[B]) \to R_1[Z_1] \rightleftharpoons R_2[Z_2]$.

MD_6: If $\phi_1 = \bigwedge_{i\in[1,\,k]}(R_1[A_i] \approx_i R_2[B_i]) \to R_1[E_1E_2] \rightleftharpoons R_2[FF]$,
then $\mathsf{LHS}(\phi_1) \rightarrowtail R_1[E_1] = R_1[E_2]$;
If $\phi_2 = \bigwedge_{i\in[1,\,k]}(R_1[A_i] \approx_i R_2[B_i]) \to R_1[EE] \rightleftharpoons R_2[F_1F_2]$,
then $\mathsf{LHS}(\phi_2) \rightarrowtail R_2[F_1] = R_2[F_2]$.

MD_7: Let $L = \bigwedge_{i\in[1,\,k]}(R_1[A_i] \approx_i R_2[B_i])$.
If $L \rightarrowtail R_1[E_1] \approx R_2[F_1]$ and $L \to R_1[E_2] \rightleftharpoons R_2[F_1]$,
then $L \rightarrowtail R_1[E_1] \approx R_1[E_2]$;
If $L \rightarrowtail R_1[E_1] \approx R_2[F_1]$ and $L \to R_1[E_1] \rightleftharpoons R_2[F_2]$,
then $L \rightarrowtail R_2[F_1] \approx R_2[F_2]$.

MD_8: Let $\phi = \bigwedge_{i\in[1,\,k]}(R_1[A_i] \approx_i R_2[B_i]) \to R_1[E_1] \rightleftharpoons R_2[F_1]$.
If ϕ and $\mathsf{LHS}(\phi) \rightarrowtail R_1[E_1] \approx R_1[E_2]$,
then (1) $\mathsf{LHS}(\phi) \to R_1[E_2] \rightleftharpoons R_2[F_1]$ if $\approx$ is $=$,
and (2) $\mathsf{LHS}(\phi) \rightarrowtail R_1[E_2] \approx R_2[F_1]$ otherwise;
If ϕ and $\mathsf{LHS}(\phi) \rightarrowtail R_2[F_1] \approx R_2[F_2]$,
then (1) $\mathsf{LHS}(\phi) \to R_1[E_1] \rightleftharpoons R_2[F_2]$ if $\approx$ is $=$,
and (2) $\mathsf{LHS}(\phi) \rightarrowtail R_1[E_1] \approx R_2[F_2]$ otherwise.

MD_9: Let $L = \bigwedge_{i\in[1,\,k]}(R_1[A_i] \approx_i R_2[B_i])$.
If $L \rightarrowtail R_1[E_1] = R_1[E_2]$ and $L \rightarrowtail R_1[E_1] \approx R_2[F_1]$,
then $L \rightarrowtail R_1[E_2] \approx R_2[F_1]$;
If $L \rightarrowtail R_2[F_1] = R_2[F_2]$ and $L \rightarrowtail R_1[E_1] \approx R_2[F_1]$,
then $L \rightarrowtail R_1[E_1] \approx R_2[F_2]$.

Fig. 5 Inference system $\mathcal{I}$ for MDs

An inference system

Using MDs and weak MDs, we propose the inference system $\mathcal{I}$ in Fig. 5. It consists of nine axioms MD_1–MD_9.

- MD_1 reveals the "symmetricity" of MDs: the order of relations R_1 and R_2 in an MD can be swapped.
- $\mathsf{MD}_2, \mathsf{MD}_3$ and MD_4 extend the reflexivity, augmentation and transitivity rules of the Armstrong's axioms for FDs, respectively
- MD_5 states that a similarity predicate $\approx$ in the LHS of an MD can be *upgraded* to equality $=$ since $\approx$ subsumes $=$.
- $\mathsf{MD}_6, \mathsf{MD}_7$ and MD_8 characterize the interactions between the matching operator and similarity predicates, in particular equality; note that MD_6 and MD_7 derive weak MDs, while MD_8 deduces standard MDs. Observe that MD_6 (MD_7) and MD_8 show the need for weak MDs of the forms φ_2 and φ_1, respectively.
- MD_9 reveals the interaction between similarity predicates and equality. It derives weak MDs from weak MDs.

Given a set Σ of MDs and another MD φ, we use $\Sigma \vdash_{\mathcal{I}} \varphi$ to denote that φ is provable from Σ using rules in $\mathcal{I}$.

Example 11 Consider Σ_c consisting of $\varphi_1, \varphi_2, \varphi_3$ of Example 2, and rck_4 of Example 5, where

$\varphi_1 :=$ credit[LN] $=$ billing[LN] $\wedge$ credit[addr] $=$ billing[post] $\wedge$
credit[FN] $\approx_d$ billing[FN] $\to$ credit[Y_c] $\rightleftharpoons$ billing[Y_b],
φ_2 : credit[tel] $=$ billing[phn] $\to$ credit[addr] $\rightleftharpoons$ billing[post],
φ_3 : credit[email] $=$ billing[email] $\to$
credit[FN, LN] $\rightleftharpoons$ billing[FN, LN], and
rck_4 : ((email, email, $=$), (tel, phn, $=$)).

Then the process to show $\Sigma_c \vdash_{\mathcal{I}} \mathsf{rck}_4$ is as follows.

(a) credit[tel] $=$ billing[phn] $\wedge$ credit[email] $=$ billing[email]
$\to$ credit[addr, email] $\rightleftharpoons$ billing[post, email]
(ψ_1, by applying MD_3 to φ_2)
(b) credit[addr] $=$ billing[post] $\wedge$ credit[email] $=$ billing[email]
$\to$ credit[addr, FN, LN] $\rightleftharpoons$ billing[post, FN, LN]
(ψ_2, by applying MD_3 to φ_3)
(c) credit[tel] $=$ billing[phn] $\wedge$ credit[email] $=$ billing[email]
$\to$ credit[addr, FN, LN] $\rightleftharpoons$ billing[post, FN, LN]
(ψ_3, by applying MD_4 to ψ_1 and ψ_2)
(d) credit[LN] $=$ billing[LN] $\wedge$ credit[addr] $=$ billing[post] $\wedge$
credit[FN] $=$ billing[FN] $\to$ credit[Y_c] $\rightleftharpoons$ billing[Y_b]
(ψ_4, by applying MD_5 to φ_1)
(e) credit[tel] $=$ billing[phn] $\wedge$ credit[email] $=$ billing[email]
$\to$ credit[Y_c] $\rightleftharpoons$ billing[Y_b]
(rck_4, by applying MD_4 to ψ_3 and ψ_4)

Similarly, rck_1, rck_2 and rck_3 of Example 5 can be deduced from Σ_c as well. □

The inference system $\mathcal{I}$ is sound and complete for the deduction analysis of MDs. That is, for any set Σ of MDs and another MD φ, $\Sigma \models_m \varphi$ iff $\Sigma \vdash_{\mathcal{I}} \varphi$, when the generic reasoning mechanism defined in Section 4.1 is concerned. That is, it only assumes *the generic axioms* given there for similarity predicates and for equality, regardless of other properties of various domain-specific similarity predicates.

Theorem 1 *The inference system $\mathcal{I}$ is sound and complete for the deduction analysis of* MDs.

In the rest of the section, we prove Theorem 1. More specifically, we show that $\mathcal{I}$ is (a) sound and (b) complete for MDs in Lemmas 1 and 2, respectively.

Lemma 1 *Rules* MD_1 – MD_9 *in the inference system* $\mathcal{I}$ *are sound for the deduction analysis of* MDs*.*

Proof We show that for any set Σ of MDs and another MD φ over R_1 and R_2, if $\Sigma \vdash_{\mathcal{I}} \varphi$, then $\Sigma \models_m \varphi$. That is, for any instance $D = (I_1, I_2)$ of (R_1, R_2) and for each *stable* instance $D' = (I'_1, I'_2)$ of D for Σ, if $(D, D') \models \Sigma$ then $(D, D') \models \varphi$.

It suffices to show that each rule in $\mathcal{I}$ is correct, which corresponds to a single step in an inference process. For if it holds, then an induction on the length of proofs using $\mathcal{I}$ can readily verify that $\mathcal{I}$ is sound.

MD_1: Let $\phi = \bigwedge_{i\in[1,\,k]}(R_1[A_i] \approx_i R_2[B_i]) \to R_1[Z_1] \rightleftharpoons R_2[Z_2]$ and $\phi^r = \bigwedge_{i\in[1,\,k]}(R_2[B_i] \approx_i R_1[A_i]) \to R_2[Z_2] \rightleftharpoons R_1[Z_1]$. If $(D, D') \models \phi$, then obviously $(D, D') \models \phi^r$.

MD_2: This rule extends the reflexivity rule of Armstrong's axioms, by distinguishing two cases: one for equality and the other for non-equality similarity predicates. The correctness follows from the definitions of MDs and weak MDs.

MD_3: This is an extension of the augmentation rule of Armstrong's axioms. That is, one can augment $\mathsf{LHS}(\phi)$ with additional similarity test $R_1[A] \approx R_2[B]$. In particular, if $\approx$ is equality $=$, then $\mathsf{RHS}(\phi)$ can also be expanded accordingly. In contrast to their FD counterpart, the augmentation axioms for MDs have to treat equality and the other similarity predicates separately. The correctness of these rules again follows from the definitions of MDs and weak MDs.

MD_4: This is the transitivity rule for MDs. To see that it is sound, consider a pair (D, D') of instances such that (a) $(D, D') \models \phi_1$, (b) for each $j \in [1, g]$, $(D, D') \models \mathsf{LHS}(\phi_1) \rightarrowtail R_1[E_j] \approx_j R_2[F_j]$, (c) $(D, D') \models \phi_2$, and (d) D' is a stable instance for the given (weak) MDs.

For any two tuples $(t_1, t_2) \in D$, if they match $\mathsf{LHS}(\phi_1)$ and L, then in D', $t_1[W_1] = t_2[W_2]$ by (a), and $t_1[E_j] \approx_j t_2[F_j]$ by (b). In addition, (t_1, t_2) match $\mathsf{LHS}(\phi_2)$ in D' by (d). From these it follows that $t_1[Z_1] = t_2[Z_2]$ in D' by (c) and (d), and hence, $\mathsf{LHS}(\phi_1) \wedge L \to \mathsf{RHS}(\phi_2)$.

MD_5: Consider instances (D, D') such that $(D, D') \models \phi$. For any tuples $(t_1, t_2) \in D$, if $t_1[A] = t_2[B]$, then $t_1[A] \approx t_2[B]$ in D. From this it follows that if $(D, D') \models \phi$, then $(D, D') \models L \bigwedge (R_1[A] = R_2[B]) \to R_1[Z_1] \rightleftharpoons R_2[Z_2]$.

MD_6: Consider instances (D, D') such that $(D, D') \models \phi_1$. For any tuples $(t_1, t_2) \in D$, if they match the LHS of ϕ_1, then $t_1[E_1E_2] = t_2[FF]$ in D'. Therefore, $t_1[E_1] = t_1[E_2]$. Hence if $(D, D') \models \phi_1$, then $(D, D') \models \mathsf{LHS}(\phi_1) \rightarrowtail R_1[E_1] = R_1[E_2]$.

Similarly, if $(D, D') \models \phi_2$, then $(D, D') \models \mathsf{LHS}(\phi_2) \rightarrowtail R_2[F_1] = R_2[F_2]$.

MD_7, MD_8 *and* MD_9: The soundness of these rules can be verified along the same lines as for MD_6. □

Lemma 2 *Rules* MD_1–MD_9 *in the inference system* $\mathcal{I}$ *are complete for the deduction analysis of* MDs*.*

Proof Sketch: We show that for a set Σ of MDs and a single MD $\varphi = \bigwedge_{i\in[1,k]}(R_1[A_i] \approx_i R_2[B_i]) \to R_1[Z_1] \rightleftharpoons R_2[Z_2]$, if $\Sigma \models \varphi$, then $\Sigma \vdash_{\mathcal{I}} \varphi$. That is, if $\Sigma \models_m \varphi$, then φ can be derived from Σ by using the rules in $\mathcal{I}$.

The proof consists of two parts. (1) We first develop a *chase* procedure to compute the *closure* $(\Sigma, \varphi)^+$ of MDs. The closure is a set of triples of the form $(R_1[A], R_2[B], \rightleftharpoons)$ (or $(R[A], R'[B], \approx)$), where R, R' are in $\{R_1, R_2\}$, such that $\Sigma \models_m \mathsf{LHS}(\varphi) \to R_1[A] \rightleftharpoons R_2[B]$ (or $\Sigma \models_m \mathsf{LHS}(\varphi) \rightarrowtail R[A] \approx R'[B]$). (2) We then show that if $(R_1[E_j], R_2[F_j], \rightleftharpoons)$ is in $(\Sigma, \varphi)^+$ for all $j \in [1, m]$, then $\Sigma \vdash_{\mathcal{I}} \mathsf{LHS}(\varphi) \to R_1[Z_1] \rightleftharpoons R_2[Z_2]$, where $Z_1 = [E_1, \ldots, E_m]$ and $Z_2 = [F_1, \ldots, F_m]$. From these it readily follows that $\mathcal{I}$ is complete.

We encourage interested readers to check proof details in the Appendix. □

Remark There are subtle differences between the inference of MDs defined on different relations and its counterpart for MDs on a single relation. For instance, in a single relation, one could write $R[A] = R[A] \to R[C] \rightleftharpoons R[D]$, which assures that for any tuple t of R, $t[C]$ and $t[D]$ have to be identical. In contrast, one cannot express this in terms of MDs across different relations.

5 An algorithm for deduction analysis

We next focus on the deduction problem for matching dependencies. The main result of this section is the following:

Theorem 2 *There exists an algorithm that, given as input a set* Σ *of* MDs *and another* MD φ *over schemas* (R_1, R_2)*, determines whether or not* $\Sigma \models_m \varphi$ *in* $O(n^2 + h^3)$ *time, where n is the size of the input* Σ *and* φ*, and h is the total number of distinct attributes appearing in* Σ *or* φ*.*

The algorithm is in quadratic-time in the size of the input when (R_1, R_2) are fixed. Indeed, h is no larger than the arity of (R_1, R_2) (the total number of attributes in (R_1, R_2)) and is often much smaller than the input size n, measured by the total number of symbols in $\Sigma \cup \{\varphi\}$. It should be remarked that the deduction analysis of MDs is carried out at compile time. That is, the analysis is performed only once, no matter how many times the record matching process is conducted. More importantly, the analysis does not involve data relations, and the size n of $\Sigma \cup \{\varphi\}$ is *much smaller* than data relations on which record matching is performed.

Compared to the $O(n)$-time complexity of FD implication, Theorem 2 tells us that although the expressive power of MDs is not for free, it does not come at too big a price.

Below we prove Theorem 2 by first developing the algorithm and then verifying the correctness of the algorithm. In the next section, we shall leverage the algorithm when computing a set of high-quality RCKs.

Overview. To simplify the discussion, we consider *w.l.o.g.* a normal form of MDs. We consider MDs ϕ of the form:

$$\bigwedge_{j\in[1,k]} (R_1[A_j] \approx_j R_2[B_j]) \to (R_1[E] \rightleftharpoons R_2[F]),$$

i.e., $\mathsf{RHS}(\phi)$ is a single pair of attributes in (R_1, R_2). This does not lose generality as an MD ψ of the general form, i.e., when $\mathsf{RHS}(\psi)$ is $R_1[Z_1] \rightleftharpoons R_2[Z_2]$, is equivalent to a set of MDs in the normal form, one for each pair of attributes in (Z_1, Z_2), by rules MD_2, MD_3 and MD_4 in the inference system $\mathcal{I}$.

In particular, we assume that the input MD φ is:

$$\varphi = \bigwedge_{i\in[1,m]} (R_1[C_i] \approx_i R_2[D_i]) \to R_1[G] \rightleftharpoons R_2[H].$$

The algorithm, referred to as MDClosure, takes MDs Σ and φ as input, and computes the *closure* of Σ and $\mathsf{LHS}(\varphi)$. The closure is the set of all pairs $(R_1[G'], R_2[H'])$ such that $\Sigma \models_m \mathsf{LHS}(\varphi) \to R_1[G'] \rightleftharpoons R_2[H']$ (see also the proof of Lemma 2). Thus one can conclude that $\Sigma \models_m \varphi$ if and only if $(R_1[G], R_2[H])$ is in the closure.

The closure of Σ and φ is stored in an $h \times h \times p$ array M. The first two dimensions are indexed by distinct attributes appearing in Σ or φ, and the last one by distinct similarity predicates in Σ or φ (including $=$). Note that $p \le |\Theta|$, where the set Θ of similarity predicates is *fixed*. In practice, p is a *constant*: in any application domain only a small set of *predefined* similarity predicates is used.

The algorithm computes M based on Σ and $\mathsf{LHS}(\varphi)$ such that for relation schemas R, R' and for similarity predicate $\approx$, $M(R[E], R'[F], \approx) = 1$ iff $\Sigma \models_m \mathsf{LHS}(\varphi) \to R[E] \approx R'[F]$. Here we use weak MDs to express intermediate results during the computation, i.e., we allow R and R' to be the same relation (either R_1 or R_2), and $\approx$ to appear in the RHS of MDs. As shown by rules MD_6, MD_7, MD_8 and MD_9 in the inference system $\mathcal{I}$, this may happen due to the interaction between the matching operator and similarity predicates.

Putting these together, algorithm MDClosure takes Σ and φ as input, computes the closure of Σ and $\mathsf{LHS}(\varphi)$ using M, and concludes that $\Sigma \models_m \varphi$ iff $M(R_1[G], R_2[H], =)$ is 1. By the inference system $\mathcal{I}$, we can set $M(R_1[G], R_2[H], =) = 1$ iff $R_1[G] \rightleftharpoons R_2[H]$ is deduced from Σ and $\mathsf{LHS}(\varphi)$.

Algorithm. Algorithm MDClosure is given in Fig. 6. While the algorithm is along the same lines as its counterpart for FD implication [1], it is more involved. Indeed, MD deduction has to deal with intriguing interactions between the matching operator and similarity predicates. Below we first present procedures for handling the interactions.

```
Algorithm MDClosure
Input:  A set $\Sigma$ of MDs and another MD $\varphi$, where
        $\mathsf{LHS}(\varphi) = \bigwedge_{i\in[1,m]}(R_1[C_i]\approx_i R_2[D_i])$.
Output: The closure of $\Sigma$ and $\mathsf{LHS}(\varphi)$, stored in array $M$.

1.  All entries of $M$ are initialized to 0;
2.  for each $i \in [1,m]$ do
3.    if AssignVal ($M, R_1[C_i], R_2[D_i], \approx_i$)
4.    then Propagate ($M, R_1[C_i], R_2[D_i], \approx_i$);
5.  repeat until no further changes
6.    for each MD $\phi$ in $\Sigma$ do
      /* $\phi = \bigwedge_{j\in[1,k]}(R_1[A_j]\approx_j R_2[B_j]) \to R_1[E] \rightleftharpoons R_2[F]$ */
7.      if there is $d \in [1,k]$ such that $M(R_1[A_d], R_2[B_d], =) = 0$
          and $M(R_1[A_d], R_2[B_d], \approx_d) = 0$
8.      then continue ;
9.      else {$\Sigma := \Sigma \setminus \{\phi\}$;
10.           if AssignVal ($M, R_1[E], R_2[F], =$)
11.           then Propagate ($M, R_1[E], R_2[F], =$);}
12. return $M$.

Procedure AssignVal ($M, R[A], R'[B], \approx$)
Input: Array $M$ with new similar pair $R[A] \approx R'[B]$.
Output: Update $M$, return true if $M$ is updated and false otherwise.

1. if $M(R[A], R'[B], =) = 0$ and $M(R[A], R'[B], \approx) = 0$
2. then {$M(R[A], R'[B], \approx) := 1$; $M(R'[B], R[A], \approx) := 1$;
3.       return true;}
4. else return false;
```

Fig. 6 Algorithm MDClosure

Procedure AssignVal. As shown in Fig. 6, this procedure takes a similar pair $R[A] \approx R'[B]$ as input. It checks whether or not $M(R[A], R'[B], \approx)$ or $M(R[A], R'[B], =)$ is already set to 1 (line 1). If not, it sets both $M(R[A], R'[B], \approx)$ and its symmetric entry $M(R'[B], R[A], \approx)$ to 1, and returns true (lines 2–3). Otherwise it returns false (line 4).

Observe that if $M(R[A], R'[B], =)$ is 1, then no change is needed, since from $R[A] = R'[B]$ it follows that $R[A] \approx R'[B]$. Indeed, the generic axioms for similarity predicates tell us that each similarity relation $\approx$ subsumes $=$.

Procedures Propagate *and* Infer. When $M(R[A], R'[B], \approx)$ is changed to 1, the change may have to be propagated to other M entries. Indeed, by the generic axioms for similarity predicates. we have the following:

(1) for each $R[E] = R[A]$ (resp. $R'[E] = R[A]$), it follows that $R[E] \approx R'[B]$ (resp. $R'[E] \approx R'[B]$). Hence entries $M(R[E], R'[B], \approx)$ (resp. $M(R'[E], R'[B], \approx)$) should also be set to 1; similarly for $R[E] = R'[B]$.
(2) If $\approx$ is $=$, then for each $R[E] \approx_d R[A]$ (resp. $R'[E] \approx_d R[A]$), we have that $R[E] \approx_d R'[B]$ (resp. $R'[E] \approx_d R'[B]$); and hence, $M(R[E], R'[B], \approx_d)$ (resp. $M(R'[E], R'[B], \approx_d)$) has to be set to 1.

```
Procedure Propagate $(M, R_1[A], R_2[B], \approx)$
Input: Array $M$ with updated similar pair $R_1[A] \approx R_2[B]$.
Output: Updated $M$ to include similarity change propagation.

1.  Q.push($R_1[A], R_2[B], \approx$);
2.  while (Q is not empty) do
3.      $(R[E], R'[E'], \approx_d)$ := Q.pop();
4.      case $(R, R')$ of
5.      (1) $R = R_1$ and $R' = R_2$
6.          Infer($Q, M, R_2[E'], R_1[E], R_1, \approx_d$);
7.          Infer($Q, M, R_1[E], R_2[E'], R_2, \approx_d$);
8.      (2) $R = R' = R_1$
9.          Infer($Q, M, R_1[E], R_1[E'], R_2, \approx_d$);
10.         Infer($Q, M, R_1[E'], R_1[E], R_2, \approx_d$);
11.     (3) $R = R' = R_2$
12.         Infer($Q, M, R_2[E], R_2[E'], R_1, \approx_d$);
13.         Infer($Q, M, R_2[E'], R_2[E], R_1, \approx_d$);

Procedure Infer$(Q, M, R[A], R'[B], R'', \approx)$
Input: Queue $Q$, array $M$, newly updated similar pair
       $R[A] \approx R'[B]$, and relation name $R''$.
Output: New similar pairs stored in $Q$ and updated $M$.

1.  for each attribute $E$ of $R''$ do
2.    if $M(R[A], R''[E], =) = 1$
3.    then {if AssignVal $(M, R'[B], R''[E], \approx)$
4.          then Q.push($R'[B], R''[E], \approx$);}
5.    if $\approx$ is $=$
6.    then for each similarity predicate $\approx_d$ $(1 \le d \le p)$ do
7.            {if $M(R[A], R''[E], \approx_d) = 1$ and
                  AssignVal$(M, R'[B], R''[E], \approx_d)$
8.             then Q.push($R'[B], R''[E], \approx_d$);}
```

Fig. 7 Procedures Propagate and Infer

In turn, these changes may trigger new changes to M, and so on. It is to handle this that procedures Propagate and Infer are used, which recursively propagate the changes.

These procedures are given in Fig. 7. They use a queue Q to keep track of and process the changes: changes are pushed into Q whenever they are encountered, and are popped off from Q and processed one by one until Q is empty.

More specifically, procedure Propagate takes a newly deduced similar pair $R[A] \approx R'[B]$ as input, and updates M accordingly. It first pushes the pair into Q (line 1). Then for each entry $R[E] \approx R'[E']$ in Q (line 3), three different cases are considered, depending on whether (R, R') are (R_1, R_2) (lines 5–7), (R_1, R_1) (lines 8–10) or (R_2, R_2) (lines 11–13). In each of these cases, procedure Infer is invoked, which modifies M entries based on the generic axioms for similarity predicates given in Section 3. The process proceeds until Q becomes empty (line 2).

Procedure Infer takes as input the queue Q, array M, a new similar pair $R[A] \approx R'[B]$, and relation R'', where R, R', R'' are either R_1 or R_2. It infers other similar pairs, pushes them into Q, and invokes procedure AssignVal to update corresponding M entries. It handles two cases, namely the cases (1) and (2) mentioned above (lines 2–4 and 5–8, respectively). The new pairs pushed into Q are processed by procedure Propagate, as described above.

Algorithm MDClosure. We are now ready to illustrate the main driver of the algorithm (Fig. 6), which works as follows. It first sets all entries of array M to 0 (line 1). Then for each pair $R_1[C_i] \approx_i R_2[D_i]$ in LHS(φ), it stores the similar pair in M (lines 2–4). After these initialization steps, the algorithm inspects each MD ϕ in Σ one by one (lines 6–11). It checks whether LHS(ϕ) is matched (line 7), and if so, it invokes procedures AssignVal and Propagate to update M based on RHS(ϕ), and propagate the changes (line 10–11). The inspection of LHS(ϕ) uses a property mentioned earlier: if $M(R_1[A], R_2[B], =) = 1$, then $R_1[A] \approx_d R_2[B]$ for any similarity predicate $\approx_d$ (line 7). Once an MD is applied, it will not be inspected again (line 9). The process proceeds until no more changes can be made to array M (line 5). Finally, the algorithm returns M (line 12).

Example 12 Recall Σ_c and rck$_4$ from Example 11. We show how rck$_4$ is deduced from Σ_c by MDClosure. We use the table below to keep track of the changes to array M after step 4 of the algorithm, when MDs in Σ_c are applied. We use c and b to denote relations credit and billing, respectively.

After step 4, M is initialized with c[email] = b[email] and c[tel] = b[phn], as given by LHS(rck$_4$). Now both LHS(φ_2) and LHS(φ_3) are matched, and thus M is updated with c[addr] $\rightleftharpoons$ b[post] (as indicated by $M(c[\text{addr}], b[\text{post}], =)$), c[FN] $\rightleftharpoons$ b[FN] and c[LN] $\rightleftharpoons$ b[LN]. As a result of the changes, LHS(φ_1) is matched, and $M(c[Y_c], b[Y_b], =)$ is set to 1. After that, no more changes can be made to array M. Since $M(c[Y_c], b[Y_b], =) = 1$, we conclude that $\Sigma \models_m$ rck$_4$.

Step	New updates to M
step 4	M(c[email], b[email], =) = M(b[email], c[email], =) = 1
	M(c[tel], b[phn], =) = M(b[phn], c[tel], =) = 1
φ_2	M(c[addr], b[post], =) = M(b[post], c[addr], =) = 1
	M(c[FN], b[FN], =) = M(b[FN], c[FN], =) = 1
φ_3	M(c[LN], b[LN], =) = M(b[LN], c[LN], =) = 1
φ_1	M(c[Y_c], b[Y_b], =) = M(b[Y_b], c[Y_c],=) = 1

As another example, we show how MDClosure deduces ψ from $\{\psi_1, \psi_2, \psi_3\}$, where ψ, ψ_1, ψ_2 and ψ_3 are:

$$\psi = R_2[A_2] = R_1[A_1] \to R_2[E_2] \rightleftharpoons R_1[B_1],$$
$$\psi_1 = R_1[A_1] \approx R_2[A_2] \to R_1[B_1] \rightleftharpoons R_2[B_2],$$
$$\psi_2 = R_1[A_1] \approx R_2[A_2] \to R_1[E_1] \rightleftharpoons R_2[B_2],$$
$$\psi_3 = R_1[A_1] \approx R_2[A_2] \to R_1[E_1] \rightleftharpoons R_2[E_2].$$

We use the table below to show how MDClosure computes array M. After step 4, M stores $R_1[A_1] = R_2[A_2]$ to reflect LHS(ψ). Then LHS(ψ_1), LHS(ψ_2) and LHS(ψ_3) are matched. Applying ψ_1 first, $R_1[B_1] \rightleftharpoons R_2[B_2]$ is added to M. Now apply ψ_2, and M is updated with $R_1[E_1] \rightleftharpoons R_2[B_2]$. Here procedure Infer($Q$, M, $R_2[B_2]$, $R_1[E_1]$, R_1) deduces a new pair $R_1[B_1] \rightleftharpoons R_1[E_1]$ from $R_1[B_1] \rightleftharpoons R_2[B_2]$ and $R_1[E_1] \rightleftharpoons R_1[B_2]$, and AssignVal is called to update M accordingly. Similarly, when ψ_3 is applied,

$R_1[E_1] \rightleftharpoons R_2[E_2]$ is added to M. When Propagate and Infer are invoked, they further infer $R_2[E_2] \rightleftharpoons R_2[B_2]$ and $R_1[B_1] \rightleftharpoons R_2[E_2]$. Accordingly, M is updated to keep track of these changes.

step	**new updates**
step 4	$M(R_1[A_1], R_2[A_2], =) = M(R_2[A_2], R_1[A_1], =) = 1$
ψ_1	$M(R_1[B_1], R_2[B_2], =) = M(R_2[B_2], R_1[B_1], =) = 1$
	$M(R_1[E_1], R_2[B_2], =) = M(R_2[B_2], R_1[E_1], =) = 1$
ψ_2	$M(R_1[E_1], R_1[B_1], =) = M(R_1[B_1], R_1[E_1], =) = 1$
	$M(R_1[E_1], R_2[E_2], =) = M(R_2[E_2], R_1[E_1], =) = 1$
ψ_3	$M(R_2[E_2], R_2[B_2], =) = M(R_2[B_2], R_2[E_2], =) = 1$
	$M(R_1[B_1], R_2[E_2], =) = M(R_2[E_2], R_1[B_1], =) = 1$

After ψ_3 is applied, M can no longer be changed. Hence $\{\psi_1, \psi_2, \psi_3\} \models_m \psi$, by $M(R_2[E_2], R_1[B_1], =) = 1$. □

Complexity analysis. MDClosure executes the **repeat** loop at most n times, since in each iteration it calls procedure Propagate, which applies at least one MD in Σ. That is, Propagate can be called at most n times *in total*. Each iteration searches at most all MDs in Σ. For the k-th call of Propagate ($1 \le k \le n$), let L_k be the number of while-loops it executes. For each loop, it takes at most $O(h)$ time since procedure Infer is in $O(h)$ time. Hence *the total cost* of updating array M is in $O((L_1 + \cdots + L_n)h)$ time. Note that $(L_1 + \cdots + L_n)$ is the total number of changes made to array M, which is bounded by $O(h^2)$. Putting these together, algorithm MDClosure is in $O(n^2 + h^3)$ time. As remarked earlier, h is usually much smaller than n, and is a constant when (R_1, R_2) are fixed. Hence, the algorithm is in $O(n^2)$ time in practice. Furthermore, it can be improved by leveraging the index structures of [8,34] for FD implication.

Proof of Theorem 2 To prove Theorem 2, it suffices to show that for any Σ and φ as described above, $\Sigma \models_m \varphi$ if and only if Algorithm MDClosure sets $M(R_1[E_1], R_2[E_2], =) = 1$. For if this holds, then by the complexity analysis given above, Algorithm MDClosure is precisely the algorithm we want. The detailed proof is deferred to the Appendix. □

6 Computing relative candidate keys

As remarked in Section 1, to improve match quality, we often need to repeat blocking, windowing and matching processes multiple times, each using a different key [15].

This gives rise to *the problem for computing* RCK*s*: given a set Σ of MDs, a pair of lists (Y_{R_1}, Y_{R_2}), and a natural number m, it is to compute a set Γ of m quality RCKs relative to (Y_{R_1}, Y_{R_2}), deduced from Σ.

This problem is non-trivial. One question concerns what metrics we should use to select RCKs. Another question is how to find m quality RCKs using the metric. One might be tempted to first compute all RCKs from Σ, sort these keys based on the metric, and then select the top m keys. This is, however, beyond reach in practice: it is known that for a single relation, there are possibly exponentially many traditional candidate keys [33]. For RCKs, unfortunately, the exponential-time complexity remains intact.

In this section we first propose a model to assess the quality of RCKs. Based on the model, we then develop an efficient algorithm to infer m RCKs from Σ. As will be verified by our experimental study, even when Σ does not contain many MDs, the algorithm is still able to find a reasonable number of RCKs. In addition, in practice it is rare to find exponentially many RCKs; indeed, the algorithm often finds the set of *all* quality RCKs when m is not very large.

Quality model. To construct the set Γ, we select RCKs based on the following criteria.

- The *diversity* of RCKs in Γ. We do not want those RCKs defined with pairs $(R_1[A], R_2[B])$ if the pairs appear frequently in RCKs that are already in Γ. That is, we want Γ to include diverse attributes so that if errors appear in some attributes, matches can still be found by comparing other attributes in the RCKs of Γ. To do this we maintain a counter $\mathsf{ct}(R_1[A], R_2[B])$ for each pair, and increase it by 1 whenever an RCK with the pair is added to Γ.
- Statistics. We consider the accuracy of each attribute pair $\mathsf{ac}(R_1[A], R_2[B])$, i.e., the confidence placed by the user in the attributes, and average lengths $\mathsf{lt}(R_1[A], R_2[B])$ of the values of each attribute pair. Intuitively, the longer $\mathsf{lt}(R_1[A], R_2[B])$ is, the more likely errors occur in the attributes; and the greater $\mathsf{ac}(R_1[A], R_2[B])$ is, the more reliable $(R_1[A], R_2[B])$ are.

Putting these together, we define the *cost* of including attributes $(R_1[A], R_2[B])$ in an RCK as:

$$\begin{aligned}\mathsf{cost}(R_1[A], R_2[B]) = w_1 \cdot \mathsf{ct}(R_1[A], R_2[B]) \\ +w_2 \cdot \mathsf{lt}(R_1[A], R_2[B]) \\ +w_3/\mathsf{ac}(R_1[A], R_2[B]),\end{aligned}$$

where w_1, w_2, w_3 are weights associated with these factors. Our algorithm selects RCKs with attributes of low cost or equivalently, high quality.

Overview. Consider RCKs $((A_1, B_1, \approx_1), \ldots, (A_k, B_k, \approx_k))$ in which for each $i \in [1, k]$, $R_1[A_i] \approx_i R_2[B_i]$ appears either in some MDs of Σ, or in the default relative key

$$\left((A'_1, B'_1, =), \ldots, \left(A'_{|Y_{R_1}|}, B'_{|Y_{R_1}|}, =\right)\right),$$

where $Y_{R_1} = \left[A'_1, \ldots, A'_{|Y_{R_1}|}\right]$ and $Y_{R_2} = \left[B'_1, \ldots, B'_{|Y_{R_1}|}\right]$. The reason for focusing on such RCKs is twofold. First, we want to preserve attribute pairs specified by MDs in Σ, which are identified as attributes that are sensible to compare either by domain experts or by learning from sample data. Second, by focusing on such RCKs one does not have to worry about weak MDs in the deduction process, and hence it reduces the computational cost. We refer to such RCKs as *normal* RCKs.

We provide an algorithm for computing RCKs, referred to as findRCKs. Given Σ, (Y_{R_1}, Y_{R_2}) and m as input, it returns a set Γ of at most m RCKs relative to (Y_{R_1}, Y_{R_2}) that are deduced from Σ. The algorithm selects RCKs defined with low-cost attribute pairs. The set Γ contains m quality RCKs if there exist at least m RCKs, and otherwise it consists of all normal RCKs deduced from Σ. The algorithm is in $O(m(l+n)^3)$ time, where l is the length $|Y_{R_1}|$ ($|Y_{R_2}|$) of Y_{R_1} (Y_{R_2}), and n is the size of Σ. In practice, m is often a *predefined* constant, and the algorithm is in *cubic-time*.

To determine whether Γ includes all normal RCKs that can be deduced from Σ, algorithm findRCKs leverages a notion of *completeness*, first studied for traditional candidate keys in [33]. To present this notion we need the following.

Consider an RCK γ and an MD ϕ. We define apply(γ, ϕ) to be the relative key γ', obtained as follows:

(1) removing all $(E, F, \approx)$ from γ if $(E, F, \rightleftharpoons)$ appears in RHS(ϕ), denoted by $\gamma \setminus \{(E, F, \approx)\}$; and
(2) adding all $(A, B, \approx)$ in LHS(ϕ) to γ, denoted by $\gamma \cup \{(A, B, \approx)\}$.

Example 13 Recall the RCK rck$_1$ from Example 5 and the MD φ_2 from Example 2:

$$\text{rck}_1 = ((\text{LN}, \text{LN}, =), (\text{addr}, \text{post}, =), (\text{FN}, \text{FN}, \approx_d))$$
$$\varphi_2 = \text{credit[tel]} = \text{billing[phn]} \rightarrow \text{credit[addr]} \rightleftharpoons \text{billing[post]}$$

Then "applying" φ_2 to rck$_1$, we get that apply(rck$_1$, φ_2) = ((LN, LN, =), (FN, FN, $\approx_d$), (tel, phn, =)). □

We are now ready to define the notion of completeness. A non-empty set Γ of RCKs is said to be *complete w.r.t.* Σ if for each normal RCK γ in Γ and each MD ϕ in Σ, there exists a RCK γ_1 in Γ such that either $\gamma_1 \preceq$ apply(γ, ϕ) or $\gamma_1 =$ apply(γ, ϕ) (recall the notion $\preceq$ from Sect. 3.2).

Intuitively, that is, for all normal RCKs that can be deduced by possible applications of MDs in Σ, they are covered by "smaller" RCKs that are already in the set Γ.

This notion of completeness allows us to check whether Γ consists of all normal RCKs deduced from Σ. As will be seen shortly, our algorithm uses the following property (see a detailed proof in the Appendix) to determine whether or not Γ needs to be further expanded. To simplify the discussion, we also include in Γ the default relative key $\left((A'_1, B'_1, =), \ldots, \left(A'_{|Y_{R_1}|}, B'_{|Y_{R_1}|}, =\right)\right)$, denoted by γ_0, where $Y_{R_1} = \left[A'_1, \ldots, A'_{|Y_{R_1}|}\right]$ and $Y_{R_2} = \left[B'_1, \ldots, B'_{|Y_{R_1}|}\right]$.

Proposition 3 *When Γ includes γ_0, Γ consists of all normal* RCK*s deduced from Σ if and only if Γ is complete w.r.t. Σ.*

```
Algorithm findRCKs
Input: Number $m$, a set $\Sigma$ of MDs, and pairwise comparable $(Y_{R_1}, Y_{R_2})$.
Output: A set $\Gamma$ of at most $m$ RCKs.

1.  $c := 0$;  $S :=$ pairing $(\Sigma, Y_{R_1}, Y_{R_2})$;
2.  let ct$(R_1[A], R_2[B]) := 0$ for each $(R_1[A], R_2[B]) \in S$;
3.  $\gamma_0 := ((A'_1, B'_1, =), \ldots, (A'_{|Y_{R_1}|}, B'_{|Y_{R_1}|}, =))$;
4.  $\Gamma := \{\gamma_0\}$;  $\Gamma' := \{\gamma_0\}$;
5.  while $\Gamma'$ is not empty do
6.     Pick an RCK $\gamma$ from $\Gamma'$;  $\Gamma' := \Gamma' \setminus \{\gamma\}$;
7.     $L_\Sigma :=$ sortMD$(\Sigma)$;
8.     for each $\phi$ in $L_\Sigma$ in the ascending order do
9.        $L_\Sigma := L_\Sigma \setminus \{\phi\}$;
10.       $\gamma' :=$ apply$(\gamma, \phi)$;   flag := true;
11.       for each $\gamma_1 \in \Gamma$ do
12.          flag := flag and $(\gamma_1 \not\preceq \gamma')$;
13.       if flag then
14.          $\gamma' :=$ minimize $(\gamma', \Sigma)$;  $\Gamma := \Gamma \cup \{\gamma'\}$;  $\Gamma' := \Gamma' \cup \{\gamma'\}$;
15.          $c := c + 1$;  incrementCt$(S, \gamma')$;  $L_\Sigma :=$ sortMD$(L_\Sigma)$;
16.          if $c = m + 1$ then return $\Gamma \setminus \{\gamma_0\}$;
17. return $\Gamma$.

Procedure minimize $(\gamma, \Sigma)$
Input: Relative key $\gamma$ and a set $\Sigma$ of MDs.
Output: An RCK.

1. $L :=$ sort $(\gamma)$;
2. for each $V = (R_1[A], R_2[B], \approx)$ in $L$ in the descending order do
3.    if $\Sigma \models_m \gamma \setminus \{V\}$      /* using algorithm MDClosure */
4.    then $\gamma := \gamma \setminus \{V\}$;
5. return $\gamma$;
```

Fig. 8 Algorithm findRCKs

Algorithm findRCKs. We are now ready to present Algorithm findRCKs, as shown in Fig. 8. Before we illustrate its details, we first present the procedures it uses.

(a) Procedure minimize takes as input Σ and a relative key γ such that $\Sigma \models_m \gamma$, where γ is not necessarily an RCK; it returns an RCK by minimizing γ. It first sorts $(R_1[A], R_2[B], \approx)$ in γ based on cost$(R_1[A], R_2[B])$ (line 1). It then processes each $(R_1[A], R_2[B], \approx)$ in the *descending* order, starting from the *most costly* one (line 2). More specifically, it *removes* $V = (R_1[A], R_2[B], \approx)$ from γ, as long as $\Sigma \models_m \gamma \setminus V$ (lines 3-4). Thus when the process terminates, it produces γ', an RCK such that $\Sigma \models_m \gamma'$. The deduction is checked by invoking algorithm MDClosure (Sect. 5).
(b) Procedure incrementCt (not shown) takes as input a set S of attribute pairs and an RCK γ. For each pair $(R_1[A], R_2[B])$ in S and γ, it increases ct$(R_1[A], R_2[B])$ by 1.
(c) Procedure sortMD (not shown) sorts MDs in Σ based on the sum of the costs of their LHS attributes. The sorted MDs are stored in a list L_Σ, in ascending order.

We now present the main driver of Algorithm findRCKs. The algorithm uses a counter c to keep track of the number of RCKs in Γ, initially set to 0 (line 1). It first collects in S all pairs $(R_1[A], R_2[B])$ that are either in (Y_{R_1}, Y_{R_2}) or in some MD of Σ (referred to as pairing$(\Sigma, Y_{R_1}, Y_{R_2})$, line 1). The counters of these pairs are set to 0 (line 2). It then adds the default relative key $\gamma_0 = (Y_{R_1}, Y_{R_2}, C)$ to Γ and Γ' (lines 3-4), where Γ' keeps the subset of RCKs in Γ that have not been processed.

After these initialization steps, findRCKs repeatedly checks whether Γ is complete *w.r.t.* Σ. If not, it expands Γ (lines 5-16). More specifically, for each unprocessed $\gamma \in \Gamma'$ and $\phi \in \Sigma$, it inspects the condition for the completeness (lines 6-11). If Γ is not complete, an RCK γ' is added to both Γ and Γ', where γ' is obtained by first applying ϕ to γ and then invoking minimize. The algorithm increases the counter c by 1, and re-sorts MDs in Σ based on the updated costs (lines 13-15).

The process proceeds until either Γ contains m RCKs (line 16; excluding the default key γ_0, which may not be a RCK), or it cannot be further expanded, i.e., Γ' is empty (line 5). In the latter case, Γ already *includes all* the normal RCKs that can be deduced from Σ (line 17), as verified by Proposition 3.

The algorithm deduces RCKs defined with attributes of low costs. Indeed, it sorts MDs in Σ based on their costs, and applies low-cost MDs first (lines 7-8). Moreover, it *dynamically adjusts* the costs after each RCK γ' is added, by increasing ct$(R_1[A], R_2[B])$ of each $(R_1[A], R_2[B])$ in γ' (lines 2, 15). Further, Procedure minimize retains attributes pairs with low costs in RCKs and removes those of high costs.

Example 14 Consider MDs Σ_c described in Example 11, and attribute lists (Y_c, Y_b) of Example 1. We illustrate how algorithm findRCKs computes a set of RCKs relative to (Y_c, Y_b) from Σ_c. We fix $m = 6$, weights $w_1 = 1$ and $w_2 = w_3 = 0$.

The table below shows how the following values are changed: (1) cost$(R_1[A], R_2[B])$ for each pair $(R_1[A], R_2[B])$ appearing in Σ_c and (Y_c, Y_b), (2) the cost of each MD in Σ_c, and (3) the set Γ of RCKs deduced. When counter $c = 0$, the table only shows these values after step 4 of the algorithm. For $c \geq 1$, the values after step 15 are given.

Attribute pairs/MDs	Counter c				
	0	1	2	3	4
cost(LN, LN)	0	1	2	2	2
cost(FN, FN)	0	1	2	2	2
cost(addr, post)	0	1	1	2	2
cost(tel, phn)	0	0	1	1	2
cost(email, email)	0	0	0	1	2
cost(Y_c, Y_b)	1	1	1	1	1
cost(LHS(φ_1))	0	3	5	6	6
cost(LHS(φ_2))	0	0	1	1	2
cost(LHS(φ_3))	0	0	0	1	2

c	New RCKs added to set Γ
0	rck$_0$: ((FN, FN, =), (LN, LN, =), (addr, post, =), (tel, phn, =), (gender, gender, =))
1	rck$_1$: ((LN, LN, =), (addr, post, =), (FN, FN, $\approx_d$))
2	rck$_2$: ((LN, LN, =), (tel, phn, =), (FN, FN, $\approx_d$))
3	rck$_3$: ((email, email, =), (addr, post, =))
4	rck$_4$: ((email, email, =), (tel, phn, =))

The algorithm deduces RCKs as follows. (a) When $c = 0$, it applies MD φ_1 to rck$_0$ and gets rck$_1$. (b) When $c = 1$, rck$_2$ is deduced by applying φ_2 to rck$_1$. (c) When $c = 2$, rck$_3$ is deduced from φ_3 and rck$_1$. (d) When $c = 3$, rck$_4$ is found by applying φ_2 to rck$_3$. (e) When $c \geq 4$, nothing is changed since no new RCKs can be found. In fact the process terminates when $c = 4$ since no more RCKs are added to Γ, and all MDs in Σ have been checked against RCKs in Γ. The final set Γ is $\{$rck$_1$, rck$_2$, rck$_3$, rck$_4\}$. Note that rck$_0$ is not returned, since it is not an RCK. In the process the MD with the lowest cost is always chosen first. □

Complexity analysis. Let l be the length of (Y_{R_1}, Y_{R_2}) and n be the size of Σ. Observe the following: (a) The outer loop (line 5) of findRCKs executes at most m iterations. (b) In each iteration, sortMD(Σ) (line 6) takes $O(n \log n)$ time. (c) The innermost loop (lines 10–11) takes $O(n|\Gamma|)$ time *in total*. (d) Procedure minimize is invoked at most m times *in total*, which in turns calls MDClosure at most $O(|\gamma|)$ times (line 13), where $|\gamma| \leq l + n$. Thus the total cost of running MDClosure is in $O(m(n + l)^3)$ time (by Theorem 2, for fixed schemas). (e) $|\Gamma| \leq m(l + n)$. Putting these together, algorithm findRCKs is in $O(m(l + n)^3)$ time.

We remark that the algorithm is efficient in practice because it is run at compile time, m is often a small constant, and n and l are much smaller than data relations.

7 Experimental evaluation

In this section, we present an experimental study of our techniques. We conducted four sets of experiments. The focus of the first set of experiments is on the scalability of algorithms findRCKs and MDClosure. Using data taken from the Web, we then evaluate the utility of RCKs in record matching. More specifically, in experiments 2 and 3 we evaluate the impacts of RCKs on the performance and accuracy of statistical and rule-based matching methods, respectively. Finally, the fourth set of experiments demonstrates the effectiveness of RCKs in blocking and windowing.

We have implemented findRCKs, MDClosure, and two matching methods: sorted neighborhood [24] and Fellegi-Sunter model [20,29] with expectation maximization (EM) algorithm for assessing parameters, in Java. The experiments were run on a machine with a Quad Core Xeon

Table 3 Measures and methods

Precision	The ratio of true duplicates correctly found by a matching algorithm to all the duplicates found
Recall	The ratio of true duplicates correctly found to all the duplicates in a dataset
s_M	The number of matched pairs with blocking (windowing)
s_U	The number of non-matched pairs with blocking (windowing)
n_M	The number of matched pairs without blocking (windowing)
n_U	The number of non-matched pairs without blocking (windowing)
PC	The completeness ratio s_M/n_M
RR	The reduction ratio $1-(s_M+s_U)/(n_M+n_U)$
FS	The Fellegi-Sunter method without using RCKs
FS_{rck}	The Fellegi-Sunter method using RCKs
SN	The Sorted Neighborhood method without using RCKs
SN_{rck}	The Sorted Neighborhood method using RCKs

(2.8GHz) CPU and 8GB of memory. Each experiment was repeated over 5 times and the average is reported.

Some measures and methods used in the experiments are summarized in Table 3.

7.1 The scalability of findRCKs and MDClosure

The first set of experiments evaluates the efficiency of algorithms findRCKs and MDClosure. Since the former makes use of the latter, we just report the results for findRCKs.

Given a set Σ of MDs, a number m, and lists (Y_{R_1}, Y_{R_2}) over schemas (R_1, R_2), algorithm findRCKs finds a set of m candidate keys relative to (Y_{R_1}, Y_{R_2}) if there exist m RCKs. We investigated the impact of the cardinality $\mathsf{card}(\Sigma)$ of Σ, the number m of RCKs, and the length $|Y_{R_1}|$ (equivalently $|Y_{R_2}|$) of Y_{R_1} on the performance of findRCKs.

The MDs used in these experiments were produced by a generator. Given schemas (R_1, R_2) and a number l, the generator randomly produces a set Σ of l MDs over the schemas.

Fixing $m = 20$, we varied $\mathsf{card}(\Sigma)$ from 200 to 2,000 in 200 increments, and studied its impact on findRCKs. The result is reported in Fig. 9(a), for $|Y_{R_1}|$ ranging over 6, 8, 10 and 12. We then fixed $\mathsf{card}(\Sigma) = 2,000$ and varied the number m of RCKs from 5 to 50 in 5 increments. We report in Fig. 9(b) the performance of findRCKs for various m and $|Y_{R_1}|$. Figures 9(a) and 9(b) tell us that findRCKs scales well with the number of MDs, the number of RCKs and the length

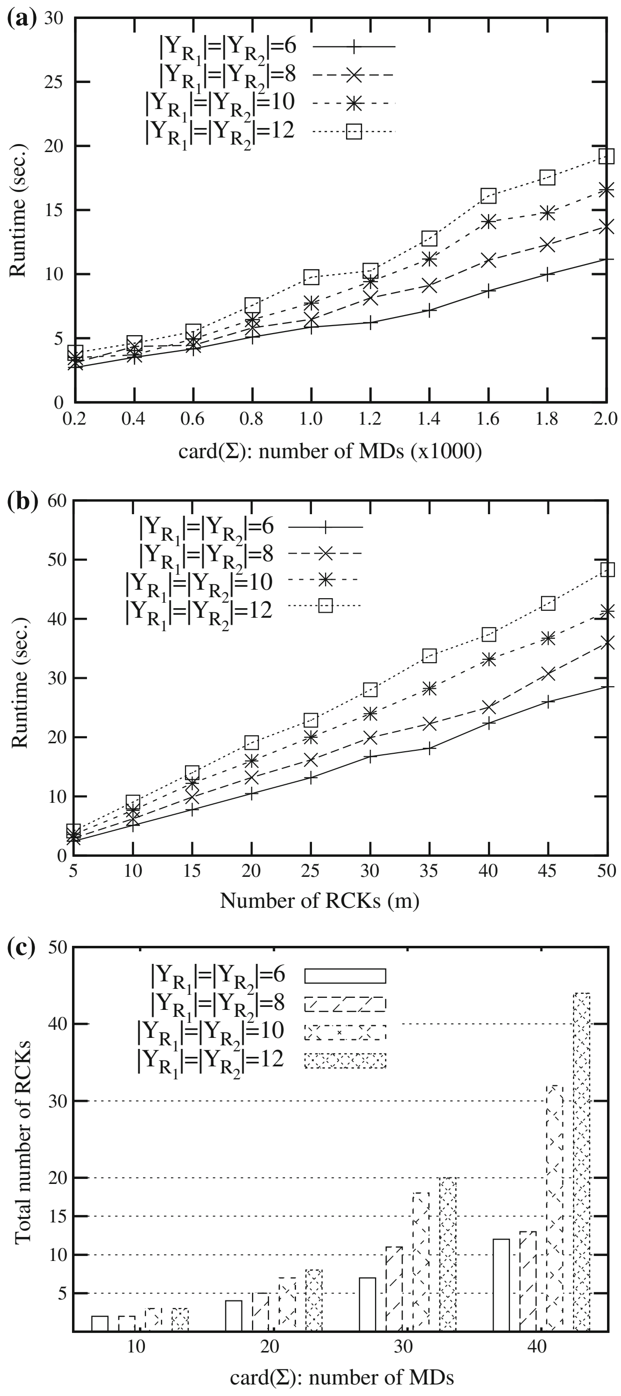

Fig. 9 Scalability of Algorithm findRCKs. **a** Scalability *w.r.t.* the number of MDs; **b** Scalability *w.r.t.* the number of RCKs; **c** The total number of RCKs

$|Y_{R_1}|$. These results also show that the larger $|Y_{R_1}|$ is, the longer it takes, as expected.

We have also inspected the quality of RCKs found by findRCKs. We find that these RCKs are reasonably diverse when the weights w_1, w_2, w_3 used in our quality model (Section 6) are 1, and $\mathsf{ac}(R_1[A], R_2[B]) = 1$ for all attribute pairs. We also used these cost parameters in the other experiments.

Figure 9(c) reports the total number of RCKs derived from small sets Σ. It shows that when there are not many MDs available, we can still find a reasonable number of RCKs that, as will be seen below, suffice to direct quality matching.

7.2 Improvement on the quality and efficiency

The next three sets of experiments focus on the effectiveness of RCKs in record matching, blocking and windowing.

Experimental setting. We used an extension of the credit and billing schemas (Section 1), also referred to as credit and billing, which have 13 and 21 attributes, respectively. We picked a pair (Y_{R_1}, Y_{R_2}) of lists over (credit, billing) for identifying card holders. Each of the lists consists of 11 attributes for name, phone, street, city, county, zip, etc. The experiments used 7 simple MDs over credit and billing, which specify matching rules for card holders.

We populated instances of these schemas using real-life data, and introduced duplicates and noises to the instances. We evaluated the ability of our MD-based techniques to identify the duplicates. More specifically, we scraped addresses in the US from the Web, and sale items (books, DVDs) from online stores. Using the data, we generated datasets controlled by the number K of credit and billing tuples, ranging from 10k to 80k. We then added 80% of duplicates, by copying existing tuples and changing some of their attributes that are not in Y_{R_1} or Y_{R_2}. Then more errors were introduced to each attribute in the duplicates, including those in Y_{R_1} and Y_{R_2}, with probability 80%, ranging from small typographical changes to complete change of the attribute.

We used the DL metric (Damerau-Levenshtein) [21] for similarity test, defined as the minimum number of single-character insertions, deletions and substitutions required to transform a value v to another value v'. We used the implementation $\asymp_\theta$ of the DL-metric provided by SimMetrics (http://www.dcs.shef.ac.uk/~sam/simmetrics.html). For any values v and v', $v \asymp_\theta v'$ if the DL distance between v and v' is no more than $(1-\theta)\%$ of $\max(|v|, |v'|)$. In all the experiments, we fixed $\theta = 0.8$.

To measure the quality of matches, we used (a) *precision*, the ratio of *true matches* (true positive) correctly found by a matching algorithm to all the duplicates found, and (b) *recall*, the ratio of true matches correctly found to all the duplicates in the dataset.

To measure the benefits of blocking (windowing), we use s_M and s_U to denote the number of matched and non-matched pairs with blocking (windowing), and similarly, n_M and n_U for matched and non-matched pairs without blocking (windowing). We then define the *pairs completeness ratio* PC and the *reduction ratio* RR as follows:

$$\mathsf{PC} = s_M/n_M, \qquad \mathsf{RR} = 1 - (s_M + s_U)/(n_M + n_U).$$

Intuitively, the larger PC is, the more effective the blocking (windowing) strategy is. In addition, RR indicates the saving in comparison space.

As the noises and duplicates in the datasets were introduced by the generator, precision, recall, PC and RR can be accurately computed from the results of matching, blocking and windowing by checking the truth held by the generator.

Experiments 2 and 3 employed windowing to improve efficiency, with a fixed window size of 10 (i.e., the sliding window contained no more than 10 tuples). The same set of windowing keys was used in all these experiments to assure that the evaluation was fair.

Exp-2: Fellegi-Sunter method (FS) [20]. This statistical method is widely used to process, e.g., census data. This set of experiments used FS to find matches, based on two comparison vectors: (a) one was the union of top five RCKs derived by our algorithms and (b) the other was picked by an expectation maximization algorithm on a sample of at most 30k tuples. The EM algorithm is a powerful tool to automatically estimate parameters such as weights and threshold [29]. We evaluated the performance of FS using these vectors, denoted by FS and $\mathsf{FS}_{\mathsf{rck}}$, respectively.

Accuracy. Figures 10(a) and (b) report the accuracy of FS and $\mathsf{FS}_{\mathsf{rck}}$, when the number K of tuples ranged from 10k to 80k. The results tell us that $\mathsf{FS}_{\mathsf{rck}}$ performs better than FS in precision, by 20% when $\mathsf{K} = 80$k. Furthermore, $\mathsf{FS}_{\mathsf{rck}}$ is less sensitive to the size of the data: while the precision of FS decreases when K gets larger, $\mathsf{FS}_{\mathsf{rck}}$ does not. Observe that $\mathsf{FS}_{\mathsf{rck}}$ and FS have almost the same recalls. This shows that RCKs effectively improve the precision (increasing the number of true positive matches) without lowering the recall (without increasing the number of false positive matches).

In these experiments, we also found that a single RCK tended to yield a lower recall, because any noise in the RCK attributes might lead to a miss-match. This is mediated by using the union of several RCKs, such that miss-matches by some RCKs could be rectified by the others. We found that $\mathsf{FS}_{\mathsf{rck}}$ became far less sensitive to noises when the union of RCKs was used.

To further evaluate the accuracy of $\mathsf{FS}_{\mathsf{rck}}$ and FS, we have conducted another experiment on a simple real-life dataset to which duplications and noises are not manually introduced. The dataset consists of 864 restaurant records with 5 attributes (name, addr, city, phone and type) taken from the Fodor's and Zagat's restaurant guides [28]. It has been known that this dataset contains 112 duplicates. When we applied the FS algorithm to the restaurant records, we used all the 5 attributes as the input and let the algorithm decide how important each attribute was when matching those records. In contrast, when we employed $\mathsf{FS}_{\mathsf{rck}}$, we let $\mathsf{FS}_{\mathsf{rck}}$ compute an RCK first and then used the RCK as the input. In this

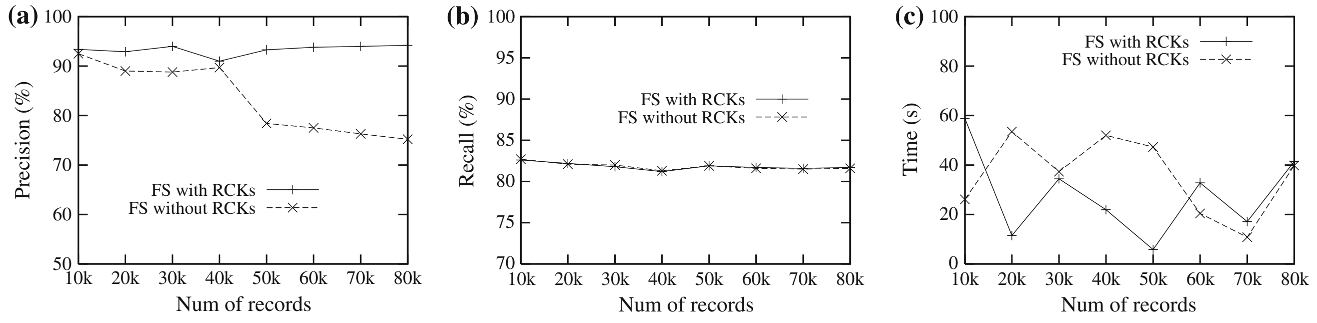

Fig. 10 Fellegi-Sunter method; **a** precision; **b** recall; **c** run time

Table 4 Fellegi-Sunter method on restaurant data

Method	Precision (%)	Recall (%)	True positive	False positive	True negative	False negative
FS	13.0	100	112	752	14341/371952	0
$\mathsf{FS}_{\mathsf{rck}}$	54.6	100	112	93	15000/372611	0
Phone	61.6	97.3	109	68	15025/372636	3

experiment, the RCK computed in $\mathsf{FS}_{\mathsf{rck}}$ is $((\mathsf{name}, \mathsf{name}, \approx_d), (\mathsf{phone}, \mathsf{phone}, \approx_d))$. As shown in Table 4, $\mathsf{FS}_{\mathsf{rck}}$ outperforms FS by 41.6% in precision. Indeed, both of them found all the 112 true matches (true positives). However, FS reported 752 false matches (false positives) while $\mathsf{FS}_{\mathsf{rck}}$ had 93 false matches. Here each number pair in the "true negative" column shows the number of non-match record pairs correctly decided by an algorithm ($\mathsf{FS}_{\mathsf{rck}}$ or FS) including or excluding the ones declared in the windowing step. Observe that using name and phone alone leads to better accuracy than using all the attributes. This is consistent with the dataset owner's suggestion that phone attribute makes the matching much easier. One may think of only using phone only for matching. However, as revealed by Table 4, it results in missing matches (see the third row). These experimental results further verify the effectiveness of using RCKs in improving the accuracy of FS.

Efficiency. As shown in Fig. 10(c), $\mathsf{FS}_{\mathsf{rck}}$ and FS have comparable performance when the number K of tuples gets large (no less than 60k). That is, RCKs do not incur extra cost while they may substantially improve the accuracy.

Exp-3: Sorted Neighborhood method (SN) **[24].** This is a popular rule-based method, which uses (a) rules of equational theory to guide how records should be compared, and (b) a sliding window to improve the efficiency. However, the quality of rule-based methods highly depends on the skills of domain experts to get a good set of rules. We run SN on the same dataset as the one used in Fig. 10 of Exp-2, based on two sets of rules: (a) the 25 rules used in [24], denoted by SN; (b) the union of top five RCKs derived by our algorithms, denoted by $\mathsf{SN}_{\mathsf{rck}}$.

Accuracy. The results on match quality are reported in Figs. 11(a) and 11(b), which show that $\mathsf{SN}_{\mathsf{rck}}$ consistently outperforms SN in both precision and recall, by around 20%. Observe that the precision of SN slightly decreases when K increases. In contrast, $\mathsf{SN}_{\mathsf{rck}}$ is less sensitive to the size of the data when precision and recall are concerned.

Efficiency. As shown in Fig. 11(c), $\mathsf{SN}_{\mathsf{rck}}$ consistently performs better than SN. This shows that RCKs effectively reduce comparisons (the number of attributes compared, and the number of rules applied), without decreasing the accuracy. Furthermore, the results tell us that both $\mathsf{SN}_{\mathsf{rck}}$ and SN scale well with the size of dataset.

The main reason that $\mathsf{SN}_{\mathsf{rck}}$ outperforms SN is as follows. Matching rules found by domain experts are often either too restrictive (with excessive attributes or unnecessary comparisons) or too relaxed (with insufficiently many attributes). Many rules of [24] have the form "if conditions A and B hold then LIKELY-MATCH; if LIKELY-MATCH and condition C holds then MATCH". Experts sometimes either overlook interrelated conditions or add unnecessary conditions. In contrast, the deduction of RCKs is able to guarantee that all the attributes in RCKs are both sufficient and necessary. An example of the rules deduced by our algorithm is $((\mathsf{SSN}, \mathsf{SSN}, =),(\mathsf{STATE}, \mathsf{STATE}, =))$, which says that if two people are in the same STATE and have identical SSN, then they are the same person. In the rule set of [24], the one closest to this is: if two people have similar SSNs and if their streets, cities and states are all pairwise similar, then the two can be identified. The rule of [24] is more restrictive than the one deduced by our algorithm, and hence, it often misses true matches.

Exp-4: Blocking and windowing. To evaluate the effectiveness of RCKs in blocking, we conducted experiments using the same dataset as before, and based on two blocking keys. One key consists of three attributes in top two RCKs derived by

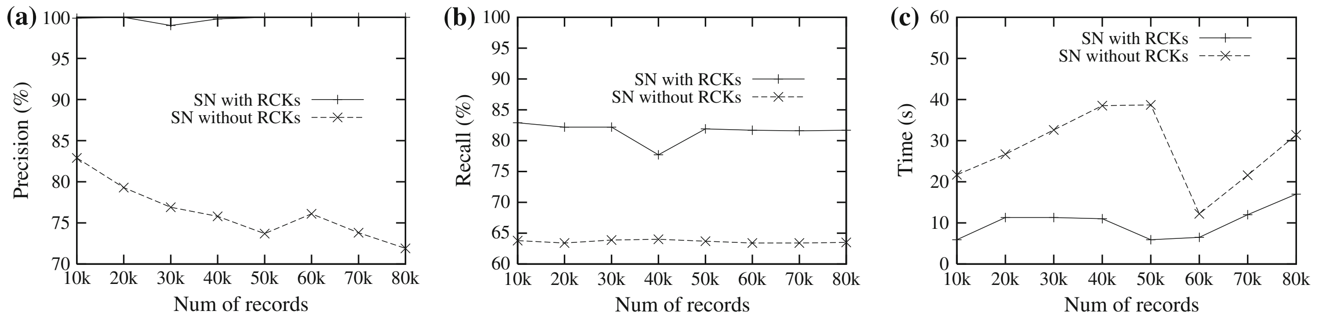

Fig. 11 Sorted Neighborhood method; **a** precision; **b** recall; **c** run time

our algorithms. The other contains three attributes manually chosen. In both cases, one of the attributes is name, encoded by Soundex [41] before blocking. Blocking keys and windowing keys (which will be discussed soon) are along the same lines as sorting keys in [25], which consist of attributes or substrings within the attributes.

The results for pairs completeness PC and reduction ratios RR are shown in Fig. 12(a) and Fig. 12(b), respectively (recall that the PC and RR can be computed by referencing the truth held by the data generator, without relying on any particular matching method). The results tell us that blocking keys based on partially encoded attributes in RCKs often yield comparable reduction ratios; at the same time, they lead to substantially better pairs completeness. Indeed, the improvement is consistently above 10%.

We also conducted experiments to evaluate the effectiveness of RCKs in windowing, and found results comparable to those reported in Fig. 12(a) and Fig. 12(b).

Summary. From the experimental results, we find the following: (a) Algorithms findRCKs and MDClosure scale well and are efficient. It takes no more than 50 seconds to deduce 50 quality RCKs from a set of 2000 MDs. (b) RCKs improve both the precision and recall of the matches found by FS and SN, and in most cases, improve the efficiency as well. For instance, it outperforms SN by around 20% in both precision and recall, and up to 30% in performance. Furthermore, using RCKs as comparison vectors, FS and SN become less sensitive to noises. (c) Using partially encoded RCK attributes as blocking or windowing keys consistently improves the accuracy of matches found.

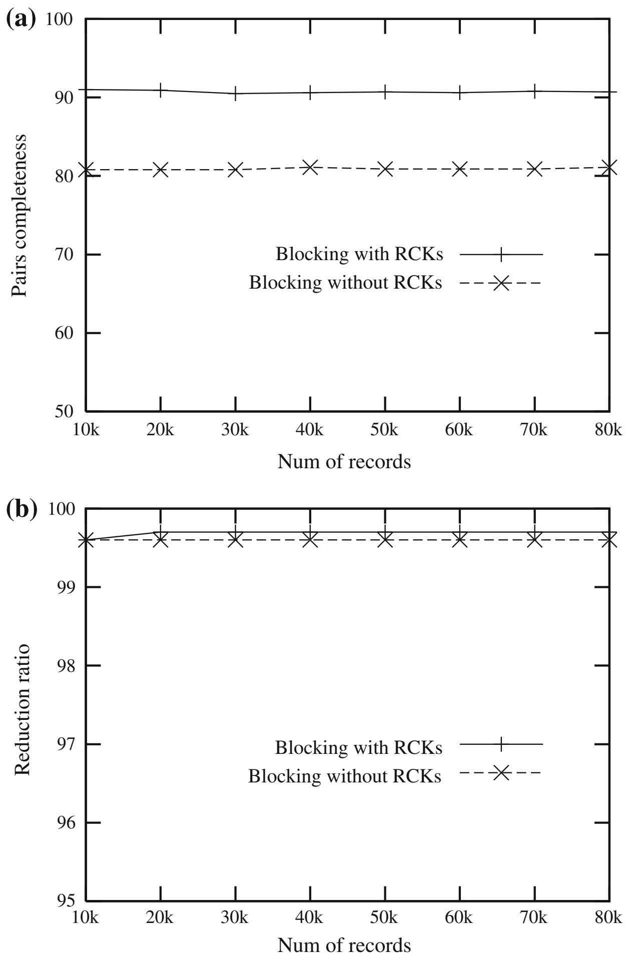

Fig. 12 Blocking; **a** Fellegi-Sunter method; **b** sorted neighborhood method

8 Conclusion

We have introduced a class of matching dependencies (MDs) and a notion of RCKs for record matching. As opposed to traditional dependencies, MDs and RCKs have a dynamic semantics and are defined in terms of similarity predicates, to accommodate errors and different representations in unreliable data sources. To reason about MDs, we have proposed a deduction mechanism to capture their dynamic semantics, a departure from the traditional notion of implication. We have also provided a sound and complete inference system and efficient algorithms for deducing MDs and quality RCKs, for matching, blocking and windowing. Our conclusion is that the techniques are a promising tool for improving match quality and efficiency, as verified by our experimental study.

Several extensions are targeted for future work. First, an extension of MDs is to support "negation", to specify when records *cannot* be matched. Second, one can augment similarity relations with constants, to capture domain-specific synonym rules along the same lines as [2,4]. Third, we have so far focused on 1-1 correspondences between attributes, as commonly assumed for record matching after data standardization [15]. As observed in [14], complex matches may involve correspondences between multiple attributes of one schema and one or more attributes of another. We are extending MDs to deal with such structural heterogeneity. Fourth, we are investigating, experimentally and analytically, the impact of different similarity metrics on match quality, and the impact of various quality models on deducing RCKs. Finally, an important and practical topic is to develop algorithms for discovering MDs from sample data, along the same lines as discovery of FDs. As remarked earlier, probabilistic methods such as EM algorithms [29,46] suggests an effective approach to discovering MDs.

Acknowledgments Fan and Ma are supported in part by EPSRC E029213/1. Li is supported in part by NGFR 973 2006CB303000 and NSFC 60533110.

Appendix: Proofs

Proof of Lemma 2

We show that for a set Σ of MDs and a single MD $\varphi = \bigwedge_{i\in[1,k]}(R_1[A_i] \approx_i R_2[B_i]) \rightarrow R_1[Z_1] \rightleftharpoons R_2[Z_2]$, if $\Sigma \models \varphi$, then $\Sigma \vdash_{\mathcal{I}} \varphi$. That is, if $\Sigma \models_m \varphi$, then φ can be derived from Σ by using the rules in $\mathcal{I}$.

The proof consists of two parts. (1) We first develop a *chase* procedure to compute the *closure* $(\Sigma, \varphi)^+$ of MDs. The closure is a set of triples of the form $(R_1[A], R_2[B], \rightleftharpoons)$ (or $(R[A], R'[B], \approx)$), where R, R' are in $\{R_1, R_2\}$, such that $\Sigma \models_m \mathsf{LHS}(\varphi) \rightarrow R_1[A] \rightleftharpoons R_2[B]$ (or $\Sigma \models_m \mathsf{LHS}(\varphi) \rightarrowtail R[A] \approx R'[B]$). (2) We then show that if $(R_1[G_j], R_2[H_j], \rightleftharpoons)$ is in $(\Sigma,\varphi)^+$ for all $j \in [1,m]$, then $\Sigma \vdash_{\mathcal{I}} \mathsf{LHS}(\varphi) \rightarrow R_1[Z_1] \rightleftharpoons R_2[Z_2]$ where $Z_1 = [G_1, \ldots, G_m]$ and $Z_2 = [H_1, \ldots, H_m]$. From these it readily follows that $\mathcal{I}$ is complete.

(1) Chase. In the first part of the proof, we start with the chase process for MDs, by extending its counterpart for traditional dependencies (see, e.g., [1]). We then show that the chase process captures MD deduction.
To simplify the exposition we use the notations below:

- We use $(\Sigma,\varphi)^+ \models_m (R[A], R'[B], \mathsf{op})$ to denote that $(R[A], R'[B], \mathsf{op})$ is in $(\Sigma,\varphi)^+$, where op is either the matching operator $\rightleftharpoons$ or a similarity predicate in Θ;
- Given MD $\phi = \bigwedge_{j\in[1,m]}(R_1[C_j] \approx_j R_2[D_j]) \rightarrow R_1[V_1] \rightleftharpoons R_2[V_2]$, we say $(\Sigma,\varphi)^+ \models_m \mathsf{LHS}(\phi)$ if and only if for each $j \in [1,m]$,
  1. $(\Sigma,\varphi)^+ \models_m (R_1[C_j], R_2[D_j], \rightleftharpoons)$ if $\approx_j$ is $=$, and
  2. $(\Sigma,\varphi)^+ \models_m (R_1[C_j], R_2[D_j], \approx_j)$ otherwise.

The chase process consists of seven steps as follows.

Step 1. Arrange the relations R_1 and R_2 in the MDs of Σ such that they are in the same order as in the MD φ.
Step 2. Initialize $(\Sigma,\varphi)^+$ with an *empty* set. For each pair $(R_1[A_i], R_2[B_i])$ $(i \in [1,k])$ in the LHS of φ, let

- $(\Sigma,\varphi)^+ = (\Sigma,\varphi)^+ \cup \{(R_1[A_i], R_2[B_i], \rightleftharpoons)\}$ if $\approx_i$ is $=$; and
- $(\Sigma,\varphi)^+ = (\Sigma,\varphi)^+ \cup \{(R_1[A_i], R_2[B_i], \approx_i)\}$ otherwise.

Step 3. For each MD $\phi = \bigwedge_{j\in[1,m]}(R_1[C_j] \approx_j R_2[D_j]) \rightarrow R_1[V_1] \rightleftharpoons R_2[V_2]$ in Σ, if $(\Sigma,\varphi)^+ \models_m \mathsf{LHS}(\phi)$, then let $(\Sigma,\varphi)^+ = (\Sigma,\varphi)^+ \cup \{(R_1[V_1[j]], R_2[V_2[j]], \rightleftharpoons)\}$ for each $j \in [1, |V_1|]$.
Step 4. (a) If $(\Sigma,\varphi)^+ \models_m (R_1[E_1], R_2[F], \mathsf{op})$ and $(\Sigma,\varphi)^+ \models_m (R_1[E_2], R_2[F], \rightleftharpoons)$, then let

- $(\Sigma,\varphi)^+ = (\Sigma,\varphi)^+ \cup \{(R_1[E_1], R_1[E_2], =)\}$, if op is $\rightleftharpoons$; and
- $(\Sigma,\varphi)^+ = (\Sigma,\varphi)^+ \cup \{(R_1[E_1], R_1[E_2], \mathsf{op})\}$ otherwise.

(b) similarly, if $(\Sigma,\varphi)^+ \models_m (R_1[C], R_2[F_1], \mathsf{op})$ and $(\Sigma,\varphi)^+ \models_m (R_1[C], R_2[F_2], \rightleftharpoons)$, then let

- $(\Sigma,\varphi)^+ = (\Sigma,\varphi)^+ \cup \{(R_2[F_1], R_2[F_2], =)\}$ if op is $\rightleftharpoons$; and
- $(\Sigma,\varphi)^+ = (\Sigma,\varphi)^+ \cup \{(R_2[F_1], R_2[F_2], \mathsf{op})\}$ otherwise.

Step 5. (a) If $(\Sigma,\varphi)^+ \models_m (R_1[E_1], R_2[F_1], \rightleftharpoons)$ and $(\Sigma,\varphi)^+ \models_m (R_1[E_1], R_1[E_2], \approx)$, then let

- $(\Sigma,\varphi)^+ = (\Sigma,\varphi)^+ \cup \{(R_1[E_2], R_2[F_1], \rightleftharpoons)\}$ if $\approx$ is $=$; and
- $(\Sigma,\varphi)^+ = (\Sigma,\varphi)^+ \cup \{(R_1[E_2], R_2[F_1], \approx)\}$ otherwise.

(b) If $(\Sigma,\varphi)^+ \models_m (R_1[E_1], R_2[F_1], \rightleftharpoons)$ and $(\Sigma,\varphi)^+ \models_m (R_2[F_1], R_2[F_2], \approx)$, then let

- $(\Sigma,\varphi)^+ = (\Sigma,\varphi)^+ \cup \{(R_1[E_1], R_2[F_2], \rightleftharpoons)\}$ if $\approx$ is $=$; and
- $(\Sigma,\varphi)^+ = (\Sigma,\varphi)^+ \cup \{(R_1[E_1], R_2[F_2], \approx)\}$ otherwise.

Step 6. (a) If $(\Sigma, \varphi)^+ \models_m (R_1[E_1], R_2[F_1], \approx)$ such that $\approx$ is not $=$ and $(\Sigma, \varphi)^+ \models_m (R_1[E_1], R_1[E_2], =)$, then let $(\Sigma, \varphi)^+ = (\Sigma, \varphi)^+ \cup \{(R_1[E_2], R_2[F_1], \approx)\}$.
(b) If $(\Sigma, \varphi)^+ \models_m (R_1[E_1], R_2[F_1], \approx)$ such that $\approx$ is not $=$ and $(\Sigma, \varphi)^+ \models_m (R_2[F_1], R_2[F_2], =)$, then let $(\Sigma, \varphi)^+ = (\Sigma, \varphi)^+ \cup \{(R_1[E_1], R_2[F_2], \mathsf{op})\}$.
Step 7. Repeat steps 3, 4, 5 and 6 until no further changes can be made to the *closure* $(\Sigma, \varphi)^+$.

Termination. Given a set $\Sigma \cup \{\varphi\}$ of MDs, the chase process given above always terminates. Indeed, it stops after making at most $(|\Theta| + 1) * h^2$ changes to $(\Sigma, \varphi)^+$ because of the following. First, the number h of attributes appearing in $\Sigma \cup \{\varphi\}$ is bounded by the total number of attributes in relations R_1 and R_2. Thus there are in total h^2 attribute pairs. Second, there are at most $|\Theta| + 1$ operators. Third, the number of elements in the closure $(\Sigma, \varphi)^+$ is bounded by $(|\Theta| + 1) * h^2$.

Chase Property. We now show that if $\Sigma \models_m \varphi$, then for each $j \in [1, m]$, $(\Sigma, \varphi)^+ \models_m (R_1[G_j], R_2[H_j], \rightleftharpoons)$, denoted by $(\Sigma, \varphi)^+ \models_m R_1[Z_1] \rightleftharpoons R_2[Z_2]$ where $Z_1 = [G_1, \ldots, G_m]$ and $Z_2 = [H_1, \ldots, H_m]$.

We prove this by contradictions. Assume that $\Sigma \models_m \varphi$, but $(\Sigma, \varphi)^+ \not\models_m R_1[Z_1] \rightleftharpoons R_2[Z_2]$. That is, there exists $j \in [1, m]$ such that $(\Sigma, \varphi)^+ \not\models_m R_1[G_j] \rightleftharpoons R_2[H_j]$. Let G_j be G and H_j be H. We construct a pair (D, D') of instances based on $(\Sigma, \varphi)^+$ such that D' is a stable instance for Σ, $(D, D') \models \Sigma$ but $(D, D') \not\models \varphi$. This contradicts the assumption that $\Sigma \models_m \varphi$.

We construct such (D, D') based on a *small model property* of MDs. That is, if $\Sigma \not\models_m \varphi$, then there exists a two-tuple instance $D = (I_1, I_2)$ of (R_1, R_2), where I_1 (resp. I_2) consists of a single tuple t_1 (resp. t_2), such that there exists a stable instance D', $(D, D') \models \Sigma$, but $(D, D') \not\models \varphi$. This property is easy to verify. In light of this, we shall construct D and D' consisting of two tuples only.

We now give the construction of (D, D'). We first group attributes and assign a unique constant to each group of attributes. We then build (D, D') from these attribute groups.

(1) *Grouping attributes*. We group the attributes by defining an equivalence relation. Let $\mathsf{attr}(R)$ denote the set of attributes in a relation schema R. For any attributes A, B in $\mathsf{attr}(R_1) \cup \mathsf{attr}(R_2)$, we say that A and B are *equivalent* if either $(A, B, =)$ or $(A, B, \rightleftharpoons)$ is in the closure $(\Sigma, \varphi)^+$.

We compute the equivalence classes as follows.

– For each attribute A in $\mathsf{attr}(R_1) \cup \mathsf{attr}(R_2)$, create an equivalent class consisting of itself only, where $\mathsf{attr}(R)$ denotes the set of all the attributes in R. We use EQ to represent all those equivalent classes, and eq_A to represent the equivalent class that attribute A belongs to.
– For any attributes A and B in $\mathsf{attr}(R_1) \cup \mathsf{attr}(R_2)$, do the following.
  – If $(A, B, =)$ or $(A, B, \rightleftharpoons)$ is in the closure $(\Sigma, \varphi)^+$, then merge eq_A and eq_B. That is, we let $\mathsf{EQ} = (\mathsf{EQ} \setminus \{eq_A, eq_B\}) \cup \{eq_{AB}\}$, where $eq_{AB} = eq_A \cup eq_B$.
  – If $(A, B, \approx)$ is in the closure $(\Sigma, \varphi)^+$, where $\approx$ is not equality, then mark $eq_A \approx eq_B$.

For each equivalent class $eq \in \mathsf{EQ}$, we assign a constant c, denoted by $eq.c$, such that for two distinct equivalent classes eq_1 and eq_2 in EQ, (a) $eq_1.c \neq eq_2.c$, (b) $eq_1.c \approx eq_2.c$ if $eq_1 \approx eq_2$, and (c) $eq_1.c \not\approx eq_2.c$ if $eq_1 \not\approx eq_2$. It is possible to find such constants since we consider dense similarity predicates (see Sect. 4.1).

(2) *Instance construction*. Based on the equivalence classes, we construct the pair (D, D') of instances as follows.
  – Let t_1 be a tuple of relation R_1 such that $t_1[A] = eq_A.c$ for each attribute $A \in \mathsf{attr}(R_1)$.
  – Let t_2 be a tuple of relation R_2 such that $t_2[B] = eq_B.c$ for each attribute $B \in \mathsf{attr}(R_2)$.
  – Let I_1 (resp. I_2) be an instance of relation R_1 (resp. R_2) consisting of the tuple t_1 (resp. t_2) only.
  – Finally, let $D = (I_1, I_2)$, and $D' = D$.

(3) *Verification*. We show that (D, D) constructed above are indeed a counter example. That is, if $\Sigma \models_m \varphi$ but $(\Sigma, \varphi)^+ \not\models_m R_1[A] \rightleftharpoons R_2[B]$, then $(D, D) \models \Sigma$, but $(D, D) \not\models \varphi$. Observe that by $(D, D) \models \Sigma$, D is a stable instance for Σ.

We first show that $(D, D) \not\models \mathsf{LHS}(\varphi) \to R_1[G] \rightleftharpoons R_2[H]$, where $\mathsf{LHS}(\varphi)$ is $\bigwedge_{i \in [1,k]} (R_1[A_i] \approx_i R_2[B_i])$. Indeed, $t_1[G] \neq t_2[H]$ since eq_G and eq_H are distinct equivalence classes, by $(\Sigma, \varphi)^+ \not\models_m R_1[G] \rightleftharpoons R_2[H]$. Furthermore, for each $i \in [1, k]$, $eq_{A_i} \approx_i eq_{B_i}$ and hence, $t_1[A_i] \approx_i t_2[B_j]$ by the construction of D. As a result, $(D, D) \not\models \mathsf{LHS}(\varphi) \to R_1[G] \rightleftharpoons R_2[H]$. Hence $(D, D) \not\models \varphi$.

We then show that $(D, D) \models \Sigma$. Assume by contradiction that there exists ϕ in Σ such that $(D, D) \not\models \phi$, where ϕ is $\bigwedge_{i \in [1,m]} (R_1[E_i] \approx_i R_2[F_i]) \to R_1[W_1] \rightleftharpoons R_2[W_2]$. That is, (t_1, t_2) match $\mathsf{LHS}(\phi)$ but $t_1[W_1] \neq t_2[W_2]$. By the construction of D, if (t_1, t_2) match $\mathsf{LHS}(\phi)$, then for each $i \in [1, m]$, $(\Sigma, \varphi)^+ \models_m R_1[E_i] \rightleftharpoons R_2[F_i]$. Then by the chase process given above, for each $j \in [1, |W_1|]$, $(W_1[j], W_2[j], \rightleftharpoons)$ would have been included in $(\Sigma, \varphi)^+$, or in other words, $(\Sigma, \varphi)^+ \models_m R_1[W_1] \rightleftharpoons R_2[W_2]$. Again by the construction of D, we would have had that

$t_1[W_1] = t_2[W_2]$, which contradicts the assumption. Hence $(D, D) \models \Sigma$.

Therefore, if $\Sigma \models_m \varphi$ then $(\Sigma, \varphi)^+ \models_m \mathsf{RHS}(\varphi)$.

(2) Simulation of the chase process. We next give the second part of the proof, by showing the following. (a) If $(\Sigma, \varphi)^+ \models_m (R_1[A], R_2[B], \rightleftharpoons)$, then $\Sigma \vdash_{\mathcal{I}} \mathsf{LHS}(\varphi) \rightarrow R_1[A] \rightleftharpoons R_2[B]$. (b) If $(\Sigma, \varphi)^+ \models_m (R[A], R'[B], \approx)$, then $\Sigma \vdash_{\mathcal{I}} \mathsf{LHS}(\varphi) \rightarrowtail R[A] \approx R'[B]$ where R, R' are relations in $\{R_1, R_2\}$ and $\approx$ is not $=$ when $R \neq R'$. If this holds, then we can conclude that if $(\Sigma, \varphi)^+ \models_m R_1[Z_1] \rightleftharpoons R_2[Z_2]$, then $\Sigma \vdash_{\mathcal{I}} \varphi$.

It suffices to show that each step of the chase process is an application of certain inference rules in $\mathcal{I}$. For if it holds, then the computation of $(\Sigma, \varphi)^+$ corresponds to a proof using rules in $\mathcal{I}$. In other words, the chase process to compute $(\Sigma, \varphi)^+ \models_m R_1[Z_1] \rightleftharpoons R_2[Z_2]$ yields a proof of $\Sigma \vdash_{\mathcal{I}} \varphi$.

Step 1. This corresponds to an application of MD_1.
Step 2. It is justified by an application of MD_2.
Step 3. This corresponds to applications of MD_4 and MD_5. More specifically, since $(\Sigma, \varphi)^+ \models_m \mathsf{LHS}(\phi)$, for each $j \in [1, m]$ we can derive the following: (a) $\Sigma \vdash_{\mathcal{I}} \phi_1$ if $\approx_j$ is $=$, where ϕ_1 is $\mathsf{LHS}(\phi) \rightarrow R_1[C_j] \rightleftharpoons R_2[D_j]$; and (b) $\Sigma \vdash_{\mathcal{I}} \phi_2$ otherwise, where ϕ_2 is either $\mathsf{LHS}(\phi) \rightarrow R_1[C_j] \rightleftharpoons R_2[D_j]$ or $\mathsf{LHS}(\phi) \rightarrowtail R_1[C_j] \approx_j R_2[D_j]$. Hence by applying MD_4 to ϕ_1 and ϕ_2, we have that $\Sigma \vdash_{\mathcal{I}} \mathsf{LHS}(\varphi) \rightarrow \mathsf{RHS}(\phi)$. Note that when $\approx_j$ is not $=$ and when only $(R_1[C_j], R_2[D_j], \rightleftharpoons)$ is in $(\Sigma, \varphi)^+$, MD_5 needs to be applied to ϕ first in order to replace $\approx_j$ with $=$.
Step 4. This is justified by applications of $\mathsf{MD}_2, \mathsf{MD}_3, \mathsf{MD}_4, \mathsf{MD}_6$ and MD_7. We verify this for case (a) as follows; the proof for case (b) is similar. For case (a) we further distinguish two cases, depending on whether op is $\rightleftharpoons$ or not.

(1) When op is $\rightleftharpoons$. Since $(\Sigma, \varphi)^+ \models_m (R_1[E_1], R_2[F], \rightleftharpoons)$ and $(\Sigma, \varphi)^+ \models_m (R_1[E_2], R_2[F], \rightleftharpoons)$, we have that $\Sigma \vdash_{\mathcal{I}} \phi_1$ and $\Sigma \vdash_{\mathcal{I}} \phi_2$, where $\phi_1 = \mathsf{LHS}(\varphi) \rightarrow R_1[E_1] \rightleftharpoons R_2[F]$ and $\phi_2 = \mathsf{LHS}(\varphi) \rightarrow R_1[E_2] \rightleftharpoons R_2[F]$. We conduct deduction analysis based on $\mathcal{I}$ as follows.
   - By applying rule MD_3 to ϕ_1, we deduce that $\phi_3 = \mathsf{LHS}(\varphi) \wedge (R_1[E_1] = R_2[F]) \rightarrow R_1[E_1E_2] \rightleftharpoons R_2[FF]$.
   - By applying rules MD_2 and MD_4 to ϕ_1 and ϕ_3, we have that $\phi_4 = \mathsf{LHS}(\varphi) \rightarrow R_1[E_1E_2] \rightleftharpoons R_2[FF]$.
   - Finally, by applying MD_6 to ϕ_4, we can deduce that $\Sigma \vdash_{\mathcal{I}} \mathsf{LHS}(\varphi) \rightarrowtail R_1[E_1] = R_1[E_2]$.

(2) When op is a non-equality similarity predicate in Θ. We can derive that $\Sigma \vdash_{\mathcal{I}} \mathsf{LHS}(\varphi) \rightarrowtail R_1[E_1]$ op $R_1[E_2]$ by using a similar argument, except that we use MD_7 here instead of MD_6.

Step 5. This is justified by an application of MD_8. We prove case (a) of the step below; the proof for case (b) is similar.

Since $(\Sigma, \varphi)^+ \models_m (R_1[E_1], R_2[F_1], \rightleftharpoons)$ and $(\Sigma, \varphi)^+ \models_m (R_1[E_1], R_1[E_2], \approx)$, we have that $\Sigma \vdash_{\mathcal{I}} \phi_1$ and $\Sigma \vdash_{\mathcal{I}} \phi_2$, where $\phi_1 = \mathsf{LHS}(\varphi) \rightarrow R_1[E_1] \rightleftharpoons R_2[F_1]$ and $\phi_2 = \mathsf{LHS}(\varphi) \rightarrowtail R_1[E_1] \approx R_1[E_2]$. By applying MD_8 to ϕ_1 and ϕ_2, we can deduce $\phi_3 = \mathsf{LHS}(\varphi) \rightarrow (R_1[E_2] \rightleftharpoons R_2[F_1])$ if $\approx$ is $=$, and $\phi_4 = \mathsf{LHS}(\varphi) \rightarrowtail (R_1[E_2] \approx R_2[F_1])$ otherwise.

Step 6. This step is justified by an application of MD_9. Again we only prove case (a) of the step; the proof for case (b) is similar.

From $(\Sigma, \varphi)^+ \models_m (R_1[E_1], R_2[F_1], \approx)$ and $(\Sigma, \varphi)^+ \models_m (R_1[E_1], R_1[E_2], =)$, it follows that $\Sigma \vdash_{\mathcal{I}} \phi_1$ and $\Sigma \vdash_{\mathcal{I}} \phi_2$, where $\phi_1 = \mathsf{LHS}(\varphi) \rightarrow R_1[E_1] \approx R_2[F_1]$ and $\phi_2 = \mathsf{LHS}(\varphi) \rightarrowtail R_1[E_1] = R_1[E_2]$. By applying MD_9 to ϕ_1 and ϕ_2, we can deduce that $\phi_3 = \mathsf{LHS}(\varphi) \rightarrow (R_1[E_2] \approx R_2[F_1])$.

Putting the two parts of proofs together, we have shown the following: (a) if $\Sigma \models_m \varphi$, then $(\Sigma, \varphi)^+ \models_m R_1[Z_1] \rightleftharpoons R_2[Z_2]$, and (b) if $(\Sigma, \varphi)^+ \models_m R_1[Z_1] \rightleftharpoons R_2[Z_2]$, then $\Sigma \vdash_{\mathcal{I}} \varphi$. Hence we can conclude that if $\Sigma \models_m \varphi$, then $\Sigma \vdash_{\mathcal{I}} \varphi$, i.e., rules MD_1–MD_9 are complete for MD deduction. □

Proof of Theorem 2

We prove this by showing that Algorithm MDClosure is precisely the algorithm we want. Recall that Algorithm MDClosure is in $O(n^2 + h^3)$ time, as shown in Section 5. It suffices to show that for any Σ and φ as described above, $\Sigma \models_m \varphi$ if and only if Algorithm MDClosure sets $M(R_1[E_1], R_2[E_2], =) = 1$.

Recall the notion of $(\Sigma, \varphi)^+ \models_m (R_1[E_1], R_2[E_2], \rightleftharpoons)$ from the proof of Lemma 2. It is already shown there that $\Sigma \models_m \varphi$ if and only if $(\Sigma, \varphi)^+ \models_m (R_1[E_1], R_2[E_2], \rightleftharpoons)$. Thus to verify the correctness of Algorithm MDClosure, it suffices to show that $(\Sigma, \varphi)^+ \models_m (R_1[E_1], R_2[E_2], \rightleftharpoons)$ if and only if MDClosure sets $M(R_1[E_1], R_2[E_2], =) = 1$.

First assume that $(\Sigma, \varphi)^+ \models_m (R_1[E_1], R_2[E_2], \rightleftharpoons)$. Then one can verify that MDClosure sets $M(R_1[E_1], R_2[E_2], =)$ to 1 by an induction on the steps when $(R_1[E_1], R_2[E_2], \rightleftharpoons)$ is included in $(\Sigma, \varphi)^+$. Indeed, there is a straightforward correspondence between the steps of chase process given in the proof of Lemma 2 and the steps of MDClosure. Based on the correspondence the induction can be readily conducted.

Conversely, assume that $M(R_1[E_1], R_2[E_2], =)$ is set to 1 by MDClosure. One can show that $(R_1[E_1], R_2[E_2], \rightleftharpoons)$ is included in $(\Sigma, \varphi)^+$ by induction on the steps when MDClosure sets $M(R_1[E_1], R_2[E_2], =)$ to 1. The induction is again based on the correspondence between the steps of MDClosure and the steps of the chase process. □

Proof of Proposition 3

First assume that Γ consists of all normal RCKs that can be deduced from Σ. Then for any normal RCK γ in Γ and each MD ϕ in Σ, $\gamma_1 = \mathsf{apply}(\gamma, \phi)$ is a normal relative key. Since Γ consists of all normal RCKs, for each RCK $\gamma_2 \preceq \gamma_1$, γ_2 is in Γ. Hence Γ is complete *w.r.t.* Σ.

Conversely, assume that Γ is complete *w.r.t.* Σ. We show that for any normal RCK γ such that $\Sigma \models_m \gamma, \gamma$ can be deduced by repeated uses of the apply operator on normal RCKs in Γ and MDs in Σ. This suffices. For if it holds, then γ must be in Γ since Γ is complete.

Since $\Sigma \models_m \gamma, \gamma$ must be in the closure $(\Sigma, \gamma_0)^+$ (recall the notion of closures from the proof of Lemma 2). Since γ is normal, the deduction of γ uses only rules MD_1–MD_4 in the inference system $\mathcal{I}$. Since these rules correspond to the apply operation, γ can be deduced from normal RCKs in Γ and MDs in Σ by the apply operations. This can be verified by induction on the steps when γ is included in $(\Sigma, \gamma_0)^+$ by the chase process given in the proof of Lemma 2. □

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Boston (1995)
2. Ananthakrishna, R., Chaudhuri, S., Ganti, V.: Eliminating fuzzy duplicates in data warehouses. In: VLDB (2002)
3. Sarma, J.W.A.D., Ullman, J.: Schema design for uncertain databases. In: Proceedings of the 3rd Alberto Mendelzon Workshop on Foundations of Data Management (2009)
4. Arasu, A., Chaudhuri, S., Kaushik, R.: Transformation-based framework for record matching. In: ICDE (2008)
5. Arasu, A., Re, C., Suciu, D.: Large-scale deduplication with constraints using Dedupalog. In: ICDE (2009)
6. Aumueller, D., Do, H.H., Massmann, S., Rahm, E.: Schema and ontology matching with COMA++. In: SIGMOD (2005)
7. Batini, C., Scannapieco, M.: Data Quality: Concepts, Methodologies and Techniques. Springer, Berlin (2006)
8. Beeri, C., Bernstein, P.A.: Computational problems related to the design of normal form relational schemas. ACM Trans. Database Syst. **4**(1), 30–59 (1979)
9. Belohlávek, R., Vychodil, V.: Data tables with similarity relations: functional dependencies, complete rules and non-redundant bases. In: DASFAA, (2006)
10. Cautis, B., Abiteboul, S., Milo, T.: Reasoning about XML update constraints. In: PODS, (2007)
11. Chaudhuri, S., Chen, B.-C., Ganti, V., Kaushik, R.: Example-driven design of efficient record matching queries. In: VLDB (2007)
12. Chaudhuri, S., Sarma, A.D., Ganti, V., Kaushik, R.: Leveraging aggregate constraints for deduplication. In: SIGMOD (2007)
13. Cohen, W.W., Richman, J.: Learning to match and cluster large high-dimensional data sets for data integration. In: KDD (2002)
14. Dhamankar, R., Lee, Y., Doan, A., Halevy, A.Y., Domingos, P.: iMAP: discovering complex mappings between database schemas. In: SIGMOD (2004)
15. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007)
16. Fan, W.: Dependencies revisited for improving data quality. In: PODS (2008)
17. Fan, W., Geerts, F., Li, J., Xiong, M.: Discovering conditional functional dependencies. IEEE Trans. Knowl. Data Eng. (2010)
18. Fan, W., Jia, X., Li, J., Ma, S.: Reasoning about record matching rules. In: VLDB (2009)
19. Fellegi, Ivan, Holt, D.: A systematic approach to automatic edit and imputation. J. Am. Stat. Assoc. **71**(353), 17–35 (1976)
20. Fellegi, I., Sunter, A.B.: A theory for record linkage. J. Am. Stat. Assoc. **64**(328), 1183–1210 (1969)
21. Galhardas, H., Florescu, D., Shasha, D., Simon, E., Saita, C.: Declarative data cleaning: language, model and algorithms. In: VLDB (2001)
22. Guha, S., Koudas, N., Marathe, A., Srivastava, D.: Merging the results of approximate match operations. In: VLDB (2004)
23. Haas, L., Hernández, M., Ho, H., Popa, L., Roth, Mary: Clio grows up: from research prototype to industrial tool. In: SIGMOD (2005)
24. Hernndez, M.A., Stolfo, S.J.: The merge/purge problem for large databases. In: SIGMOD (1995)
25. Hernndez, M.A., Stolfo, S.J.: Real-world data is dirty: data cleansing and the merge/purge problem. Data Min. Knowl. Discov. **2**(1), 9–37 (1998)
26. Huhtala, Y., Kärkk ainen, J., Porkka, P., Toivonen, H.: TANE: an efficient algorithm for discovering functional and approximate dependencies. Comput. J. **42**(2), 100–111 (1999)
27. http://www.sas.com/industry/fsi/fraud/
28. http://userweb.cs.utexas.edu/users/ml/riddle/data.html
29. Jaro, M.A.: Advances in record-linkage methodology as applied to matching the 1985 census of Tampa Florida. J. Am. Stat. Assoc. **89**, 414–420 (1989)
30. Koudas, N., Saha, A., Srivastava, D., Venkatasubramanian, S.: Metric functional dependencies. In: ICDE (2009)
31. Lim, E.-P., Srivastava, J., Prabhakar, S., Richardson, J.: Entity identification in database integration. Inf. Sci. **89**(1–2), 1–38 (1996)
32. Loshin, D.: Master Data Management. Knowledge Integrity, Inc., New York (2009)
33. Lucchesi, C.L., Osborn, S.L.: Candidate keys for relations. J. Comput. Syst. Sci. **17**(2), 270–279 (1978)
34. Maier, D.: The Theory of Relational Databases. Computer Science Press, Rockville (1983)
35. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. (2001)
36. Sarawagi, S., Bhamidipaty, A.: Interactive deduplication using active learning. In: KDD (2002)
37. Sauter, G., Mathews, B., Ostic, E.: Information Service Patterns, Part 3: Data Cleansing Pattern. IBM, USA (2007)
38. Shen, W., Li, X., Doan, A.: Constraint-based entity matching. In AAAI (2005)
39. Singla, P., Domingos, P.: Object identification with attribute-mediated dependences. In: PKDD (2005)
40. Sismanis, Y., Brown, P., Haas, P.J., Reinwald, B.: GORDIAN: efficient and scalable discovery of composite keys. In: VLDB (2006)
41. Soundex: http://en.wikipedia.org/wiki/Soundex
42. Verykios, V.S., Elmagarmid, A.K., Houstis, E.: Automating the approximate record-matching process. Inf. Sci. **126**(1–4), 83–89 (2002)
43. Vianu, V.: Dynamic functional dependencies and database aging. J. ACM **34**(1), 28–59 (1987)

44. Weis, M., Naumann, F.: DogmatiX tracks down duplicates in XML. In: SIGMOD (2005)
45. Weis, M., Naumann, F., Jehle, U., Lufter, J., Schuster, H.: Industry-scale duplicate detection. In: VLDB (2008)
46. Winkler, W.E.: Methods for record linkage and bayesian networks. Technical report RRS2002/05, U.S. Census Bureau (2002)
47. Winkler, W.E.: Methods for evaluating and creating data quality. Inf. Syst. **29**(7), 531–550 (2004)
48. Yancey, W.: BigMatch: A program for extracting probable matches from a large file. Technical report computing 2007/01, U.S. Census Bureau (2007)