2-1996

Entity identification in database integration

Ee Peng LIM
Singapore Management University, eplim@smu.edu.sg

Jaideep SRIVASTAVA

Satya PRABHAKAR

James RICHARDSON

DOI: https://doi.org/10.1016/0020-0255(95)00185-9

NORTH-HOLLAND

Informatics and Computer Science

Entity Identification in Database Integration*

EE-PENG LIM

School of Applied Science, Nanyang Technological University, Nanyang Avenue, Singapore 2263

JAIDEEP SRIVASTAVA

Department of Computer Science, University of Minnesota Minneapolis, MN 55455

SATYA PRABHAKAR

and

JAMES RICHARDSON

Honeywell Sensor and System Development Center, Minneapolis, MN 55455

Communicated by C. V. Ramamoorthy

ABSTRACT

The objective of entity identification is to determine the correspondence between objective instances from more than one database. This paper examines the problem at the instance level assuming that schema level heterogeneity has been resolved a priori. Soundness and completeness are defined as the desired properties of any entity-identification technique. To achieve soundness, a set of identity and distinctness rules have to be established for the entities in the integrated world. We then propose the use of extended key, which is the union of keys (and possibly other attributes) from the relations to be matched, and its corresponding identity rule to determine the equivalence between tuples from relations that may not share any common key. Instance level functional dependencies (ILFD), a form of semantic constraint information about the real-world entities, are used to derive the missing extended key attribute values of a tuple. Formal properties of ILFDs are derived. Results from a Prolog-based prototype entity-identification system are presented.

* Supported in part by contract F30602-91-C-0128 from Rome Laboratory of the U.S. Air Force.

1. INTRODUCTION

Database integration is the problem of taking two (or more) independently developed databases and resolving the differences between them to make them appear as one. The need for integration may arise due to new applications that span multiple databases (e.g., an organization may want an application that carries out an enterprise-wide analysis of operations) or due to the integration of operations of different organizations (for example, corporate mergers and acquisitions, or integrated billing, as in the case of U.S. West and AT&T). Two kinds of integration are possible:

- *Virtual Integration*: A *virtually integrated* database is created on top of the component databases, usually by means of a common data model and integrated schema, while the components retain their identities and usage. The effort in federated autonomous databases is in this direction [14].
- *Actual Integration*: An *actually integrated* database is created from the component databases. The original databases are discarded and the applications are migrated to the new integrated database [17].

In this paper, we focus our attention on the entity identification problem that can occur in both virtual and actual database integration. In a single database context, it is usually the case that an object instance can uniquely model a real-world entity. This property does not hold for multiple autonomous databases and the problem of entity identification therefore arises. Kent described this as the breakdown of the information model [8]. For example, when we add two object instances to a relation in a single database, the one-to-one correspondence between object instances and real-world entities assures that the two new object instances refer to distinct real-world entities. However, when the two object instances are added to relations in different databases, such one-to-one correspondence property may disappear.

Pre-existing databases in most organizations are defined and populated by different people at different times in response to different organizational or end-user requirements. Such independent development of databases often results in two databases capturing parts of the same real-world domain. Typically, when there is a need to provide integrated access to these related databases, relating the representations of the same real-world entity from the two databases is often difficult, if not impossible, without specifying *additional* semantic information that resolves this ambiguity.

The rest of this paper is organized as follows. In Section 2, we describe the background of the entity-identification problem and give an example that motivates this research. We also give some brief comments on some existing approaches. In Section 3, we give a formal treatment of the problem and characterize soundness, completeness, and monotonicity as the desired properties of any entity-identification process. We then propose a new approach in Section 4 and give a formal analysis in Section 5. Our entity identification prototype is described in Section 6. Conclusions are given in Section 7.

2. BACKGROUND

The task of integrating pre-existing autonomous databases has to resolve the *logical heterogeneity* that arises when the participating databases are designed independently of one another [2]. Logical heterogeneity can occur at two levels, namely, schema level and instance level. The resolution of schema level heterogeneity is known as *schema integration.* The resolution of instance level heterogeneity is known as *instance integration.*

1. *Schema Level*: The meta-data information of the participating databases, equally applicable to all instances, are incompatible. The incompatibility problems at this level include:
   - *Domain mismatch*: The domains of similar attributes are not compatible in structure or semantics. For example, the **currency** attribute in one relation being in U.S. dollars while the corresponding **currency** attribute in another relation is in yen is a case of semantic mismatch. An example of structural mismatch is the case when the **name** attribute in one relation has a data type of string while the **name** attribute in another relation is composed of three subattributes of string data type, namely, **lastname**, **firstname**, and **middlename**.
   - *Schema mismatch*: This problem arises when the schema structures and semantics of two databases are not compatible, for example, the **Employee** table in one database may correspond to a union of **Part-time-employee** and **Full-time-employee** tables in another database.
   - *Constraint mismatch*: The constraints specified in the participating databases may be incompatible. For example, a graduate school database may have the constraint of requiring all graduate students to have a cummulative GPA of greater than 3.0, whereas the computer science department database may have the constraint of requiring all graduate students to have a cumulative GPA of greater than 3.5.

2. *Instance Level*: The schemas are compatible in structure (attribute domains) and semantics (attribute meaning), but the instances corresponding to the same real-world entity have yet to be identified and merged. The two problems that occur at this level are:
   - *Entity identification*: This is the problem of identifying object instances from different databases that correspond to the same real-world entity. Related to the entity identification problem is the *instance level homonym* problem. Instance level homonyms occur when the same identifier is used for different real-world entities in different databases [12]. The instance level homonym problem is different from the homonym problems mentioned in most literature. Homonym problems are often discussed at the attribute level where the meanings assigned to attribute names are different in two databases [3]. In general, there appears to be no fully automatic way to solve the instance level homonym problem [1].
   - *Attribute value conflict*: Attribute value conflict arises when the attribute values in the two databases, modeling the same property of a real-world entity, do not match. This conflict may be caused by *data scaling conflict*, *inconsistent data*, or *missing data* [15] or even potential schema modeling errors. Data scaling conflict occurs when the domains of semantically related attributes use different units of measurement. Inconsistent data occur when semantically equivalent attributes have different values. Missing data refers to the situation when object instances modeling the same real world do not have the same set of attributes. It is clear that attribute value conflict resolution can be performed only after the entity-identification problem has been resolved.

Schema level homonym and synonym problems are usually resolved at the schema integration stage. In the case of actual database integration, the instance level problems must be resolved subsequently to complete the integration process. In the case of virtual database integration, the strategies and information required for resolving instance level problems have to be specified during design time, i.e., schema integration phase, but the actual processing only takes place during the query time.

Resolving instance level ambiguities is a common problem existing in a federated database context. However, instance integration has not been discussed much in the literature. Most of the database integration research focuses on the schema integration problem. It is commonly believed that the instance integration can be easily performed after schema integration is completed. As we shall see in Section 2.1, it is not always easy to integrate object instances even when the schemas are compatible. In the case of federated databases, participating database systems can continue to operate autonomously. Instance integration may have to be performed whenever updating is done on the participating databases. Because entity identification is the first problem to be tackled in instance integration, effective and efficient approaches to handle it are necessary.

2.1. MOTIVATING EXAMPLE

In the following, we show an example of the entity-identification problem. Consider the relations R and S from databases DB_1 and DB_2, respectively, as shown in Table 1. Both relations contain tuples that describe restaurant entities in the real world.

EXAMPLE 1

Relation R has **(name, street)** as its candidate key, whereas relation S has **(name, city)** as its candidate key. In this paper, candidate keys in relations are underlined. To integrate relations R and S, we first have to determine which tuples in R and S, respectively, describe the same restaurant entity in the integrated world.

A popular approach of using a common candidate key for identification does not work because R and S do not share a common candidate key. The common key attribute, **name**, may suggest that the first tuple in R and the first tuple in S refer to the same restaurant entity because they have the same value, i.e., **name** = "VillageWok". Nevertheless, a careful analysis reveals that this conclusion may not be correct. For example, if we insert a tuple with **name** = "VillageWok" and **street** = "Penn.Ave." into R, we will have a situation where one tuple in S can be matched with two tuples in R. It is not clear which of them is the correct one.

On the other hand, if we were told that restaurant entities in the integrated world have unique combinations of name, street, and city attribute values, Wash.Ave. is only in city Mpls, and the restaurant owned by Hwang is only on Wash.Ave., we can safely conclude that the first tuple in R and the first tuple in S refer to the same restaurant entity. The insertion of a tuple with **name** = "VillageWok" and **street** = "Penn.Ave."

TABLE 1

R			S		
name	**street**	**cuisine**	**name**	**city**	**manager**
VillageWok	Wash.Ave.	Chinese	VillageWok	Mpls	Hwang
Ching	Co.B Rd.	Chinese	OldCountry	Roseville	Libby
OldCountry	Co.B2 Rd.	American	ExpressCafe	Burnsville	Tom

into *R* does not cause any problem because we know that it is not the restaurant owned by Hwang.

With this example, we illustrate that entity identification in general is not a trivial problem. In this paper, we investigate the use of extra semantic information to (at least partially) automate the entity-identification process.

2.2. *EXISTING APPROACHES*

The existing approaches to entity identification can be categorized as follows:

1. *Using key equivalence.* Many approaches assume some common key exists between relations from different databases modeling the same entity type, e.g., Multibase [5, 7]. Because a key can be used for uniquely associating object instance with real-world entities, equivalence of values of the common key can be used to resolve the problem. This approach, however, is limited because the relations may have no common key, even though they might share some common key attributes, as shown in Example 1.
2. *User-specified equivalence.* This approach requires the user to specify equivalence between object instances, e.g., as a table that maps local object ids to global object ids, i.e., the responsibility of matching the object instance is assigned to the user. This technique has been suggested for the Pegasus project [1]. Because the matching table can be very large, this approach can potentially be extremely cumbersome. Nevertheless, it is a general approach and can handle synonym and homonym problems.
3. *Use of probabilistic key equivalence.* Instead of insisting on full key equivalence, Pu [13] suggested matching object instances using only a portion of the key values in the restricted domain. The name matching problem, as an instance of the key equivalence, has been addressed by matching the subfields of names. If most of the subfields in two given names match, the names are considered to be identical. Although this approach can produce a high confidence on the matching result, it is applicable only when common key exists between relations. The probabilistic nature of matching may also admit erroneous matching.
4. *Use of probabilistic attribute equivalence.* Chatterjee and Segev proposed the use of all common attributes between two relations to determine entity equivalence [4]. For each pair of records from two relations, a value called *comparison value* is assigned based on a probabilistic model. Nevertheless, in Section 2.1, we demonstrate that comparing common attribute values does not necessarily produce correct matching results.

5. *Use of heuristic rules.* Wang and Madnick attacked the problem using a knowledge-based approach [18]. A set of heuristic rules is used to infer additional information about the object instances to be matched. Because the knowledge used is heuristic in nature, the matching result produced may not be correct.

From the above, we conclude that key equivalence is a well-accepted solution technique when it is applicable. Most entity-identification techniques have been proposed based on different assumptions of entity equivalence. The notion of correctness for entity-identification processes has not been well formulated. We also realized that most techniques do not handle cases when two object instances do not have a common candidate key. In this paper, we give a formal treatment on the entity-identification problem. We propose the notions of soundness and completeness as the desired properties of an entity identification process. To achieve a sound identification result, we require identity rules and distinctness rules to be established. Extra knowledge, known as *instance level functional dependency* (ILFD), is used to match tuples in two relations that share no common candidate key.

3. THE ENTITY-IDENTIFICATION PROBLEM

3.1. PROBLEM FORMULATION

The aim of entity identification is to determine the correspondence between object instances from multiple databases. To simplify the discussion, we assume that the data model used is relational and real-world entities of the same type can be represented as tuples in relations. Each relation is expected to have one or more candidate keys to uniquely identify its tuples.[1] Each key consists of one or more attributes called *key attributes*. Each tuple in a relation models some properties of a unique real-world entity. However, each real-world entity may be modeled by many tuples, provided that no two such tuples can be found in the same relation.[2] We also assume that the attribute values of tuples are accurate with respect to that of the corresponding real-world entities.[3] Two tuples from different relations are said to *match* if they model the same real-world

[1] If no key is defined, the entire attribute set of the relation can be treated as the key.

[2] This assumption is often satisfied by relations in the existing databases.

[3] Only the attribute values that are consistent with properties of the real-world entities can participate in the entity-identification process.

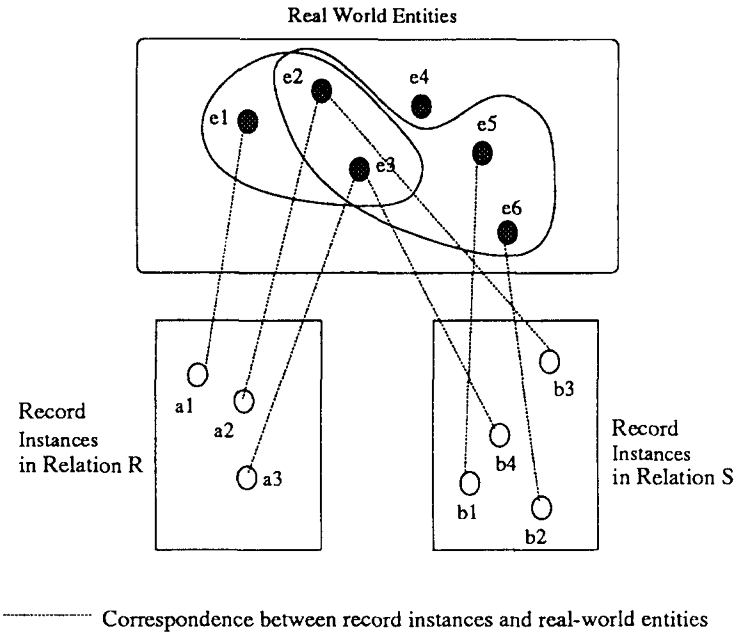
Real World Entities
e1
e2
e3
e4
e5
e6
Record Instances in Relation R
a1
a2
a3
Record Instances in Relation S
b1
b2
b3
b4
Correspondence between record instances and real-world entities

Fig. 1. Relationship between real-world entities and tuples.

entity, as illustrated by Figure 1. Relations R and S contain tuples that represent a set of real-world entities. Because some real-world entities may not be modeled in either relation, e.g., e_4, we are only interested in the subset of real-world entities modeled by at least one of R and S. This subset is known as the integrated world. In the example, a_2 and b_3 match and a_3 and b_4 match.

Related to the entity identification problems are the synonym and homonym problems. For example, different employee numbers assigned to the same employee in different relations is an example of the synonym problem, whereas the same employee number in two relations for different employees exemplifies the homonym problem. The synonym problem arises due to the fact that the attribute employee numbers in both relations are not semantically equivalent. Because semantically equivalent attributes can usually be determined at the schema integration stage [10, 19], we assume that the synonym problem would have been resolved before entity identification was performed. Although the homonym problem may arise due to semantically unequivalent attributes, it can also be caused by the fact that the key of the relation is not the key in the integrated world. For example, let R be a relation that contains tuples describing restaurant entities in Minneapolis and let S be another relation that contains tuples describing restaurant entities in St. Paul. Both relations have **name** as key. Nevertheless, **name** is not necessarily the key in the integrated set of restaurant entities because the same restaurant name may exist in both Minneapolis and St. Paul.

To differentiate between value equivalence and entity equivalence, we use $a = b$ to denote the former and $a \equiv b$ to denote the latter from now on.

3.2. SOUNDNESS AND COMPLETENESS OF THE ENTITY-IDENTIFICATION PROCESS

The entity-identification process can be expressed as a three-valued function that takes a pair of tuples and returns "true" only if they refer to the same real-world entity, "false" only if they do not, and "unknown" otherwise. Based on the function values, all pair of tuples can be partitioned into three disjoint sets, namely, identical pairs, distinct pairs, and undetermined pairs. Those pairs evaluating to "true" or "false" can be represented in a *matching table* and a *negative matching table*, respectively. Because each tuple has a unique identifier in its relation, a matching (negative matching) table entry consists of the key values of the pair of tuples. Moreover, the record pairs in the matching table and negative matching table have to satisfy the following constraints.

UNIQUENESS CONSTRAINTS. *No tuple in either relation can be matched to more than one tuple in the other relation.*

CONSISTENCY CONSTRAINT. *No tuple pair can appear in both the matching and negative matching tables.*

Consider two relations R and S, coming from different databases, both of which model real-world entities of type E. We call the conceptual matching table MT_{RS} and the conceptual negative matching table NMT_{RS}.
In the following, we define soundness and completeness of entity identification. They are the desired properties to be achieved by the entity-identification process.

DEFINITION (Soundness). Each record pair declared to be matching (not matching) indeed models the same (distinct) real-world entity.

DEFINITION (Completeness). The entity-identification process returns a value of "matching" or "not matching", but not "undetermined", for all pairs of tuples.

In reality, soundness and completeness of solutions to the entity-identification problem are difficult to achieve. Nevertheless, at least the soundness property must be achieved for an entity-identification process to be successful. In Figure 2, we illustrate a particular entity identification process that fails to satisfy the soundness property.

Figure 2 depicts a scenario in which the attribute values (including the key values) of tuples are identical, but the records model two different real-world entities. If by using attribute value equivalence we conclude that tuples r_1 and s_1 match, soundness is violated. The above entity-identification process fails because it could not recognize that database 1 and 2 are modeling different subsets of the domain of real-world entities. To differentiate between the two tuples, we include an extra attribute in each relation to indicate the domain attribute of value "DB1", i.e., r_1 = ("VillageWok","Chinese","DB1"). With the domain attribute, we can define assertions or semantic rules relevant to the entities modeled by a particular database. Note that a domain attribute may or may not be modeled in the integrated database depending on whether the source location information has to be made available to the user.

To achieve soundness, all information used for entity identification must be correct with respect to the integrated world. Moreover, some identity and distinctness rules need to be established for entities in the integrated world. These rules are asserted by the database administrator (DBA) or a collaborative group of database administrators, who has a better understanding of the integrated domain of real-world entities. Advanced techniques in knowledge discovery may also suggest some identity or distinctness rules that have been overlooked by the database

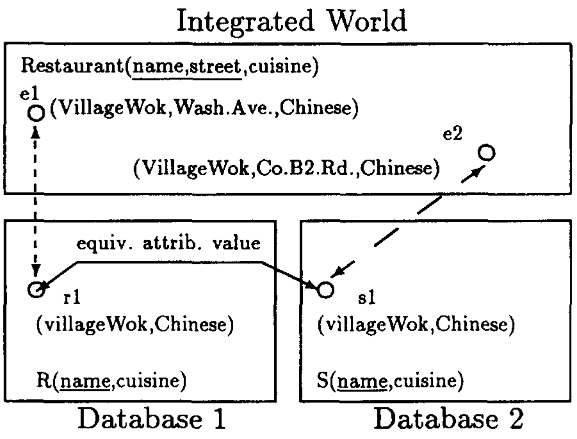

Fig. 2. Difficulties in guaranteeing soundness and completeness.

administrator. The entity-identification process should use this set of rules to determine matched and unmatched tuples.

DEFINITION (Identity rule). An identity rule for the set of real-world entities E is of the form

$$\forall e_1, e_2 \in E, \qquad P(e_1.A_1, \ldots, e_1.A_m, e_2.B_1, \ldots, e_2.B_n) \rightarrow (e_1 \equiv e_2),$$

where P is a conjunction of predicates on the attributes $A_1, \ldots, A_m$, and $B_1, \ldots, B_n$ of e_1 and e_2, respectively. Each predicate is either of the form e_i*.attribute op* e_j*.attribute* or e_j*.attribute op value*, where $op \in \{=, <, >, \leqslant, \geqslant, \neq\}$. Furthermore, for each $e_1.A_i$ or $e_2.A_i$ that appears in the predicates, P must imply $e_1.A_i = e_2.A_i$.

EXAMPLE

Let E denotes a set of restaurant entities. Consider the rules:

r1: $\forall e_1, e_2 \in E$, (e_1*.cuisine* = "Chinese") $\wedge$ (e_2*.cuisine* = "Chinese") $\rightarrow$ $(e_1 \equiv e_2)$.
r2: $\forall e_1, e_2 \in E$, (e_1*.cuisine* = "Chinese") $\rightarrow (e_1 \equiv e_2)$.

r1 is an identity rule, but r2 is not because its antecedent does not imply e_2*.cuisine* = e_1*.cuisine*.

Consider entities $e_1, e_2 \in E$. The existence of A_k as an identifying attribute can be captured by the identity rule

$$\forall e_1, e_2 \in E, \qquad (e_1.A_k = e_2.A_k) \rightarrow (e_1 \equiv e_2).$$

Suppose e_1 and e_2 were modeled in relations R_1 and R_2, respectively, and attribute A_k was used as the key in both. The above identity rule then is used by the technique called *entity identification by key equivalence*.

Similar to identity rules, a set of distinctness rules can be defined to determine unmatched tuples.

DEFINITION (Distinctness rule). A distinctness rule for the set of real-world entities E is of the form

$$\forall e_1, e_2 \in E, \qquad P(e_1.A_1, \ldots, e_1.A_m, e_2.B_1, \ldots, e_2.B_n) \rightarrow (e_1 \not\equiv e_2),$$

where P is a conjunction of predicates on the attributes $A_1, \ldots, A_m$, and $B_1, \ldots, B_n$ of e_1 and e_2. Each predicate is either of the form e_i*.attributed op* e_j*.attribute* or e_i*.attribute op value*, where $i \in \{1, 2\}$ and $op \in \{=, <, >,$

$\leqslant, \geqslant, \neq\}$. Furthermore, P must involve some attribute from each of e_1 and e_2.

EXAMPLE

r3: $\forall e_1, e_2 \in E$, $(e_1.\mathit{speciality} = \text{“Mughalai”}) \wedge (e_2.\mathit{cuisine} \neq \text{“Indian”}) \rightarrow (e_1 \neq e_2)$.

The above distinctness rule says that restaurant entity e_1 specialized in Mughalai food is not equivalent to the restaurant entity e_2 with non-Indian cuisine.

In general, it is necessary though not sufficient to enforce the identity/distinctness rules in the integrated world as constraints in the relations to be matched. For example, for the identity rule r1 to hold, we have to ensure that there is at most one Chinese restaurant in *every* relation modeling the restaurant entities. In other words, the uniqueness of tuple in a relation satisfying the identity rule conditions must be observed. The above constraint is not sufficient because it does not ensure that whenever two Chinese restaurant records are added to relations in different databases, they refer to the same real-world restaurant. A common means to achieve uniqueness for an identity rule, which contains only predicates of the form $e_1.A_i = e_2.A_i$ for $1 \leqslant i \leqslant m$, is to treat a subset of $A_1, \ldots, A_m$ that appears in the relation as key.

Similarly, for the distinctness rule r3 to hold, we have to ensure that for each relation modeling the restaurant entities, no non-Indian restaurant tuple can have specialty in Mughalai food. The above constraint is not sufficient because it is still possible to have two records across different databases both referring to the same real-world entity and both satisfying the constraints in each database, but they together violate the distinctness rule.

To guarantee completeness, we require enough information to determine whether every pair of tuples matches or not. This means that a complete set of identity (distinctness) rules, and a complete knowledge about the domain of real-world entities modeled by the relations may be needed. Such complete knowledge is often difficult, if not impossible, to obtain. To cope with incompleteness, an entity identification technique should allow the DBA to supply more information as more knowledge about the real-world is gained.

3.3. MONOTONICITY OF ENTITY IDENTIFICATION

Given a set of identity/distinctness rules, entity identification can be viewed as a reasoning process which derives the conditions required by the

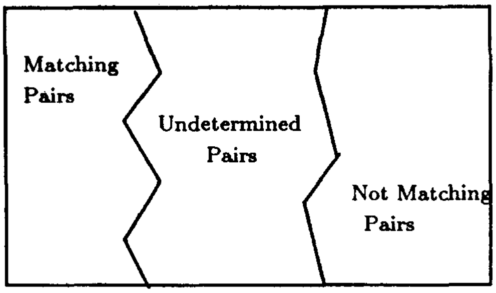

Fig. 3. Three kinds of matching relationships.

antecedents of the rules. To guarantee soundness of the entity-identification process, the technique used should be *monotonic* [11].

DEFINITION (Monotonic entity-identification technique). An entity-identification technique is monotonic if every pair of tuples determined by the technique to be matching/not matching remains so when additional information is supplied.

Pictorially, we can visualize the relationships between pairs of tuples as the Venn diagram shown in Figure 3.

If the entity-identification technique adopted is monotonic, the sets of matching pairs and non matching pairs will expand, whereas the set of undetermined pairs shrinks as more semantic information becomes available. Completeness is achieved only when the undetermined set is empty.

4. PROPOSED SOLUTION

In this section, we propose a new approach to solve the entity-identification problem. Our approach differs from previous approaches in the following aspects:

1. Our technique is developed under the assumption that a sound matching result is desired. For example, a company wanting to dismiss employees with sales performance below expectation requires matching between the employee records in one database and their performance records in another database. It is crucial that the set of matched records be correct; otherwise, some people may be wrongly fired. Our technique achieves soundness by using valid constraints about the integrated real-world to perform matching. Object instances are matched only when they

satisfy some identity rule. This is in contrast to some approaches that rely heavily on heuristics, or a probabilistic model.

2. Our technique removes the requirement for a common key between relations to be matched. This offers a more general approach toward entity identification.

3. Using a matching table to contain the result of entity identification, our technique does not exclude the use of other approaches to assert additional possible matching record pairs in the table. For example, it is possible for a knowledgeable user to add entries directly to the matching table.

We define the concept of extended key and extended key equivalence. The extended key equivalence, as a kind of identity rule, can be used with instance level functional dependencies (ILFDs) to match tuples from two relations sharing no common candidate key.

4.1. EXTENDED KEY EQUIVALENCE AND INSTANCE LEVEL FUNCTIONAL DEPENDENCIES

In Section 2.2, we mentioned that key equivalence is a common approach for entity identification by matching tuples from two relations, in the presence of a common candidate key. In the previous section, we showed that key equivalence is one kind of identity rule. An additional and often unstated assumption for key equivalence to work is that "the (common) candidate key continues to remain as a key for the unionized set of real-world entities." Key equivalence will not work when the relations to be matched do not have any common candidate key. As a result, we may have to use other kinds of identity rules to perform entity identification.

The following is an approach that uses relationships/equivalences between key (and potentially nonkey) attributes to establish entity equivalence.

Let R_1 and R_2 be relations (in different databases) that model (potential subsets of) a set of real-world entities E. Let K_1 and K_2 be the keys of R_1 and R_2, respectively. We define the concepts of *extended key* and *extended key equivalence* as follows.

DEFINITION (Extended key). The extended key (denoted by K_{Ext}) is a minimal set of attributes, of the form $K_1 \cup K_2 \cup \tilde{A}$, needed to uniquely identify an instance of type E in the integrated real world, where $\tilde{A}$ is a set of attributes of E in neither K_1 nor K_2.[4]

[4] The extended key will be used as the key of the integrated relation. If a common candidate key exists and key equivalence identity rule holds, we have $K_{\text{Ext}} = K_1 = K_2$. Quite often, we may have $K_{\text{Ext}} = K_1 \cup K_2$.

Given an extended key K_{Ext}, the corresponding identity rule, extended key equivalence is defined as follows.

DEFINITION (Extended key equivalence). By its definition, the concept of extended key gives rise to an identity rule of the form

$$\forall e_1, e_2 \in E, \qquad (e_1.A_1 = e_2.A_1) \wedge \cdots \wedge (e_1.A_k = e_2.A_k) \rightarrow (e_1 \equiv e_2),$$
$$\text{where } K_{\text{Ext}} = \{A_1, A_2, \ldots, A_k\}.$$

Extended key equivalence is an interesting identity rule in that it does not require constraints other than the key constraint to be enforced on the relations to be matched in order to guarantee that the tuples satisfying the matching condition are unique in their relations.

EXAMPLE 2

For example, in Table 2, R and S are two relations that model the real-world entity type Restaurant. Key equivalence is not applicable because R and S do not share any common key. However, the domain of restaurant entities may have the extended key $K_{\text{Ext}} = (\text{name}, \text{cuisine})$, with the corresponding extended key equivalence rule being:

$$\forall e_1, e_2 \in \textit{Restaurant}, \qquad (e_1.\textit{name} = e_2.\textit{name}) \wedge (e_1.\textit{cuisine} = e_2.\textit{cuisine})$$
$$\rightarrow (e_1 \equiv e_2).$$

Because relation S does not have the attribute *cuisine*, this rule is not directly applicable. However, if relation S could be extended to include this attribute, by using additional information perhaps, the rule could be used for entity identification. For example, if we know that every restaurant specializing in Mughalai food should be an Indian restaurant, then we can conclude that the second tuple in R matches the tuple in S, as shown in Table 3.

In this paper, we call this kind of semantic information *instance level functional dependence* (ILFD). ILFDs can be used to derive the missing key attribute values that are required for using extended key equivalence.

TABLE 2

R			S		
name	**cuisine**	**street**	**name**	**speciality**	**city**
TwinCities	Chinese	Wash.Ave.	TwinCities	Mughalai	St. Paul
TwinCities	Indian	Univ.Ave.			

TABLE 3

MT_{RS}

R.name	**R.cuisine**	**S.name**
TwinCities	Indian	TwinCities

Let relation R model (a subset of) the set of real-world entities E.

Definition (ILFD). An ILFD is a semantic constraint on the real-world entities. It is of the form

$$\forall e \in E, \qquad (e.A_1 = a_1) \wedge \cdots \wedge (e.A_n = a_n) \rightarrow (e.B = b),$$

where $A_1, \ldots, A_n$ and B are attributes (possibly including the domain attributes) and $a_1, \ldots, a_n$ and b are the possible attribute values.

We express the above ILFD as

$$(E.A_1 = a_1) \wedge \cdots \wedge (E.A_n = a_n) \rightarrow (E.B = b).$$

For example, (*Restaurant.speciality* = "Mughalai") → *Restaurant.cuisine* = "Indian" is used in Example 2, for entities of type Restaurant.

In many ways, ILFDs are very similar to the functional dependencies (FDs) used in the database design process. Both are semantic constraints on the possible relations that can be valid instances of a relation scheme. For example, given a restaurant relation R(*name*, *cuisine*, *speciality*), the FD *name* → *cuisine* means that pairs of tuples in R having identical *name* values must also have identical *cuisine* values. An ILFD *speciality* = "Sichuan" → *cuisine* = "Chinese" would mean that every tuple in R that has *speciality* as "Sichuan" must have *cuisine* as "Chinese". In fact, they look identical when the boolean conditions in ILFDs are replaced by propositional symbols.

Nevertheless, FDs and ILFDs are still different in the following ways:

- The implication sign → in an FD is read as "functionally determines," whereas the counterpart in an ILFD is the usual implication used in mathematical logic.
- The antecedent and consequent of an FD are sets of attributes, whereas the antecedent and consequent of an ILFD are sets of propositional symbols with each set denoting the conjunction of its members.
- Checking for violation of FDs involves at least two tuples whereas checking for violation of ILFDs involves only one tuple.

- FDs are typically useful in designing relation schemes that do not contain redundancy. This database design process is known as *normalization.* ILFDs are useful in deriving new properties of real-world entities from existing properties.

It is assumed that all tuples modeling in the real world are consistent with the ILFDs. Not all attributes of a real-world entity are modeled as attributes in a relation. One or more keys, each consisting of one or more attributes, are selected to uniquely identify the entity in the relation.

Moreover, ILFDs can help to determine if a pair of tuples does not model the same real-world entity. The following proposition shows that each ILFD indeed corresponds to a distinctness rule.

PROPOSITION 1. *$(E.A_1 = a_1) \wedge \cdots \wedge (E.A_n = a_n) \rightarrow (E.B = b)$ is an ILFD if and only if $\forall e_1, e_2 \in E$, $(e_1.A_1 = a_1) \wedge \cdots \wedge (e_1.A_n = a_n) \wedge (e_2.B \neq b) \rightarrow (e_1 \not\equiv e_2)$ is a distinctness rule.*

Proof. (Only if) Suppose $(E.A_1 = a_1) \wedge \cdots \wedge (E.A_n = a_n) \rightarrow (E.B = b)$ is an ILFD. Given any $e_1, e_2 \in E$,

$$(e_1 \equiv e_2) \rightarrow \big[((e_1.A_1 = a_1) \wedge \cdots \wedge (e_1.A_n = a_n)) \\ \leftrightarrow ((e_2.A_1 = a_1) \wedge \cdots \wedge (e_2.A_n = a_n))\big].$$

Applying the ILFD, we have

$$(e_1 \equiv e_2) \rightarrow \big[((e_1.A_1 = a_1) \wedge \cdots \wedge (e_1.A_n = a_n)) \leftrightarrow (e_2.B = b)\big],$$

$$(e_1 \equiv e_2) \rightarrow \big[\neg((e_1.A_1 = a_1) \wedge \cdots \wedge (e_1.A_n = a_n)) \vee (e_2.B = b)\big],$$

$$(e_1 \not\equiv e_2) \vee \neg((e_1.A_1 = a_1) \wedge \cdots \wedge (e_1.A_n = a_n)) \vee (e_2.B = b).$$

Therefore,

$$(e_1.A_1 = a_1) \wedge \cdots \wedge (e_1.A_n = a_n) \wedge (e_2.B \neq b) \rightarrow (e_1 \not\equiv e_2).$$

(If) Suppose we are given

$$\forall e_1, e_2 \in E, \qquad (e_1.A_1 = a_1) \wedge \cdots \wedge (e_1.A_n = a_n) \wedge (e_2.B \neq b) \rightarrow (e_1 \not\equiv e_2).$$

Letting $e \equiv e_1 \equiv e_2$, we get

$$\forall e \in E, \qquad (e.A_1 = a_1) \wedge \cdots \wedge (e.A_n = a_n) \wedge (e.B \neq b) \rightarrow \text{FALSE}.$$

Therefore, $(E.A_1 = a_1) \wedge \cdots \wedge (E.A_n = a_n) \rightarrow (E.B = b)$ is an ILFD. □

TABLE 4
Negative Matching NMT_{RS}

R.name	R.cuisine	S.name
TwinCities	Chinese	TwinCities

For example, the corresponding distinctness rule for the above ILFD example is

$$\forall e_1, e_2 \in \mathit{Restaurant}, \quad (e_1.\mathit{speciality} = \text{"}\mathit{Mughalai}\text{"}) \wedge (e_2.\mathit{cuisine} \neq \text{"}\mathit{Indian}\text{"}) \rightarrow (e_1 \not\equiv e_2).$$

Applying this rule to Example 2, we arrive at a pair of tuples that do not model the same real-world entity. This distinct pair is asserted in the conceptual *negative matching table* NMT_{RS} as shown in Table 4.

Because the number of nonmatching RS pairs is usually much larger than that of matching RS pairs (because the maximum of matching pairs is the minimum of R and S sizes), our approach does not try to present all nonmatching RS pairs explicitly. Instead, we keep those $R(S)$ tuples not matched with any $S(R)$ tuple as separate tuples in the integrated table, while merging the matching pairs into one. We denote the integrated table by T_{RS}. Because R and S may not have all extended key attributes, NULL values may exist in the extended key attributes of T_{RS}. As a result of this, we have to assign a new interpretation to the integrated table T_{RS}.

Within the integrated table T_{RS}, a real-world entity can be modeled by more than one tuple.[5] A T_{RS} tuple can possibly match another T_{RS} tuple provided they have no conflicting nonnull values in their extended key.

Given tables R and S, and the matching table MT_{RS}, the integrated table T_{RS} can be expressed using relational operations. Let K_R and K_S be the keys of tables R and S, respectively. The integrated table T_{RS} can be expressed as $\mathrm{MT}_{RS} \overset{\leftrightarrow}{\bowtie}_{K_R} R \overset{\leftrightarrow}{\bowtie}_{K_S} S$, where $\overset{\leftrightarrow}{\bowtie}$ denotes the full outer-join operator.

4.2. *MATCHING TABLE CONSTRUCTION*

Let R and S be two relations that model the same real-world entity type E, with K_R and K_S as their respective keys. Let K_{Ext} be an extended

[5] In this case, it is at most two.

TABLE 5

R			*S*		
name	**cuisine**	**street**	**name**	**speciality**	**county**
TwinCities	Chinese	Co.B2	TwinCities	Hunan	Roseville
TwinCities	Indian	Co.B3	TwinCities	Sichuan	Hennepin
It'sGreek	Greek	FrontAve.	It'sGreek	Gyros	Ramsey
Anjuman	Indian	LeSalleAve.	Anjuman	Mughalai	Mpls.
VillageWok	Chinese	Wash.Ave.			

key, with the missing key attributes in R and S as $K_{\text{Ext}-R}$ and $K_{\text{Ext}-S}$, respectively, i.e.,

$$K_{\text{Ext}-R} = K_{\text{Ext}} - K_R,$$

$$K_{\text{Ext}-S} = K_{\text{Ext}} - K_S.$$

The matching table, MT_{RS}, is constructed as follows:

- Extend relation R, to R', with attributes $K_{\text{Ext}-R}$ and set the missing attribute values of each tuple to be NULL. The extended relation S' is derived analogously.
- Apply the available ILFDs to derive the values for $K_{\text{Ext}-R}$ and $K_{\text{Ext}-S}$ for each R' and S' tuple.
- For each pair of R' and S' tuples that have identical nonnull values for K_{Ext}, append $(R'.K_R, S'.K_S)$ to the MT_{RS} table.

EXAMPLE 3 (Matching table construction)

Let R and S be relations that model the real-world Restaurant entities as shown in Table 5. Suppose we have the extended key {**name**, **cuisine**, **speciality**} and its corresponding extended key equivalence rule:

$$\forall e_1, e_2 \in E, \quad (e_1.name = e_2.name) \wedge (e_1.cuisine = e_2.cuisine) \wedge (e_1.speciality = e_2.speciality) \rightarrow (e_1 \equiv e_2).$$

Let the following be the available ILFDs:

I1: $(Restaurant.speciality = \text{“Hunan”}) \rightarrow (Restaurant.cuisine = \text{“Chinese”})$.

I2: $(Restaurant.speciality = \text{“Sichuan”}) \rightarrow (Restaurant.cuisine$

= "Chinese").

I3: (*Restaurant.speciality* = "Gyros") → (*Restaurant.cuisine* = "Greek").

I4: (*Restaurant.speciality* = "Mughalai") → (*Restaurant.cuisine* = "Indian").

I5: (*Restaurant.name* = "TwinCities") ∧ (*Restaurant.street* = "Co.B2") → (*Restaurant.speciality* = "Hunan").

I6: (*Restaurant.name* = "Anjuman") ∧ (*Restaurant.street* = "LeSalleAve.") → (*Restaurant.speciality* = "Mughalai").

I7: (*Restaurant.street* = "FrontAve.") → (*Restaurant.county* = "Ramsey").

I8: (*Restaurant.name* = "It'sGreek") ∧ (*Restaurant.county* = "Ramsey") → (*Restaurant.speciality* = "Gyros").

The following is a derived ILFD:

I9: (*Restaurant.name* = "It'sGreek") ∧ (*Restaurant.street* = "FrontAve.") → (*Restaurant.speciality* = "Gyros").

The extended relations R' and S' are shown in Table 6. The matching table MT_{RS} is shown in Table 7.

The number of ILFDs useful to a relation and their format heavily depend on the database domain. In some cases, there may be few ILFDs that can be useful to the database relations, and the attributes involved in their consequent and antecedents may vary widely. There are also cases in which a large number of ILFDs are useful to some relations and their formats are uniform. For the second category of useful ILFDs, it may be storage efficient to store the ILFDs are relations.[6] For example, the ILFDs I1, I2, I3, and I4 in Example 3 can be stored in a relation as shown in Table 8.

Let $IM_{(\bar{x},y)}$ denote the ILFD table that involves attributes $\bar{x}$ and derives attribute y. Given tables R and S, let R's attributes be $r_1,\ldots,r_p$ and let S's attributes be $s_1,\ldots,s_q$. Let the missing key attributes in R be $y_1,\ldots,y_m$

[6] ILFDs of the form $(E.A1 = a1) \wedge \cdots \wedge (E.An = an) \rightarrow (E.B = b)$ can be stored in the relation schema ILFD($A1, A2, \ldots, An, B$).

TABLE 6

R'				S'			
<u>**name**</u>	<u>**cuisine**</u>	<u>**speciality**</u>	**street**	<u>**name**</u>	<u>**speciality**</u>	<u>**cuisine**</u>	**county**
TwinCities	Chinese	Hunan	Co.B2	TwinCities	Hunan	Chinese	Roseville
TwinCities	Indian	NULL	Co.B3	TwinCities	Sichuan	Chinese	Hennepin
It'sGreek	Greek	Gyros	FrontAve.	It'sGreek	Gyros	Greek	Ramsey
Anjuman	Indian	Mughalai	LeSalleAve.	Anjuman	Mughalai	Indian	Mpls.
VillageWok	Chinese	NULL	Wash.Ave.				

TABLE 7
MT_{RS}

R.name	R.cuisine	S.name	S.speciality
TwinCities	Chinese	TwinCities	Hunan
It'sGreek	Greek	It'sGreek	Gyros
Anjuman	Indian	Anjuman	Mughalai

and the missing key attributes in S be $z_1, \ldots, z_n$. Given a key missing attribute y_i (z_i) in R (S), there may be several ILFD tables that can be used to derive y_i (z_i), each using different original attributes of R. Let them be $IM_{(r_i 1, y_i)}, \ldots, IM_{(r_i u, y_i)}$ ($IM_{(s_i 1, z_i)}, \ldots, IM_{(s_i v, z_i)}$).

The matching table MT_{RS} can be obtained as a series of relational expressions as shown below. Essentially, the set of relational expressions first derives the missing extended key attribute values for records in R and S. Using each $IM_{(r_i j, k_i)}$ ($IM_{(s_i j, z_i)}$), a relation named $R^j_{y_i}$ ($S^j_{z_i}$) containing the original R (S) key and the missing extended key attribute y_i (z_i) is computed. Combining all $R^j_{y_i}$s ($S^j_{z_i}$s), we obtain R_{y_i} (S_{z_i}) that contains key attributes of R (S) as well as the missing extended key attribute y_i (z_i). R (S) is subsequently extended with all y_i's (z_i's) by a series of outer joins. Finally, the extended relations R' and S' are joined over the extended key K_{Ext} to obtain the matching table MT_{RS}:

For each y_i, $1 \leqslant i \leqslant m$,

$$R^1_{y_i} = \prod_{K_{R,y_i}} \left(R \bowtie_{r_i 1} IM_{(r_i 1, y_i)} \right),$$

$$R^2_{y_i} = \prod_{K_{R,y_i}} \left(R \bowtie_{r_i 2} IM_{(r_i 2, y_i)} \right),$$

$$\vdots$$

TABLE 8
ILFD table $IM_{(speciality, cuisine)}$

speciality	cuisine
Hunan	Chinese
Sichuan	Chinese
Gyros	Greek
Mughalai	Indian

$$R^u_{y_i} = \prod_{K_{R,y_i}} \left(R \bowtie_{r_i u} IM_{(r_i u, y_i)}\right),$$

$$R_{y_i} = \bigcup_{j=1}^{u} R^j_{y_i},$$

$$R' = R \overset{\leftrightarrow}{\bowtie}_{K_R} R_{y1} \overset{\leftrightarrow}{\bowtie}_{K_R} \cdots \overset{\leftrightarrow}{\bowtie}_{K_R} R_{y_m};$$

for each z_i, $1 \leqslant i \leqslant n$,

$$S^1_{z_i} = \prod_{K_S, z_i} \left(S \bowtie_{s_i 1} IM_{(s_i 1, z_i)}\right),$$

$$S^2_{z_i} = \prod_{K_S, z_i} \left(S \bowtie_{s_i 2} IM_{(s_i 2, z_i)}\right),$$

$$\vdots$$

$$S^v_{z_i} = \prod_{K_S, z_i} \left(S \bowtie_{s_i v} IM_{(s_i v, z_i)}\right),$$

$$S_{z_i} = \bigcup_{j=1}^{v} S^j_{z_i},$$

$$S' = S \overset{\leftrightarrow}{\bowtie}_{K_S} S_{z_1} \overset{\leftrightarrow}{\bowtie}_{K_S} \cdots \overset{\leftrightarrow}{\bowtie}_{K_S} S_{z_n},$$

$$\mathrm{MT}_{RS} = \prod_{K_R, K_S} \left(R' \bowtie_{K_{\mathrm{Ext}}} S'\right).$$

Figure 4 depicts the overall entity-identification process of using ILFD tables. The entity-identification process reads in R and S relations, derives their extended key, and generates the integrated table T_{RS}.

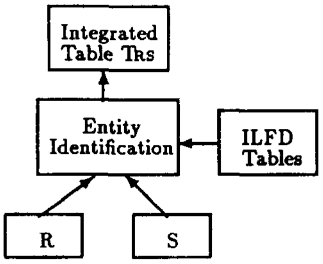

Fig. 4. Entity identification using ILFD tables.

5. FORMAL PROPERTIES OF ILFDS

In this section, we explore the properties of ILFDs and describe their relationship to *functional dependencies* (FDs). We establish an ILFD theory similar to the FD theory. The latter has been addressed in most database textbooks [16]. To simplify subsequent discussion, we adopt the notation shown in Table 9. By limiting our discussion to the set of ILFDs pertaining to a specific entity set E, we can eliminate E from the ILFD definition and represent an ILFD as

$$((A_1 = a_1) \wedge \cdots \wedge (A_m = a_m)) \rightarrow (B = b).$$

Each $(A_i = a_i)$ or $(B = b)$ can be treated as a propositional symbol. As a result, we can transform an ILFD into a formula in propositional logic shown below:

$$(P_1 \wedge \cdots \wedge P_m) \rightarrow Q.$$

In the above formula, P_i denotes $(A_i = a_i)$ and Q denotes $(B = b)$. Similar to functional dependencies, two or more ILFDs with identical antecedent conditions can be combined into one formula as follows:

$$((P_1 \wedge \cdots \wedge P_m) \rightarrow Q_1) \wedge \cdots \wedge ((P_1 \wedge \cdots \wedge P_m) \rightarrow Q_n)$$
$$\equiv (P_1 \wedge \cdots \wedge P_m) \rightarrow (Q_1 \wedge \cdots \wedge Q_n).$$

In the remainder of this paper, we assume that the information about an entity set can be represented as a relation. Each tuple in the relation represents an entity and each attribute of the tuple represents a property about the corresponding entity. Because all relations mentioned in this

TABLE 9
Notation

Symbols	Meaning
P, Q, R, etc.	Propositional symbols
X, Y, Z, etc.	Conjunction of propositional symbols
A, B, C, etc.	Attributes
a, b, c, etc.	Constants

section model sets of entities, we shall use the terms relation and entity set interchangeably.

We say that a relation R *satisfies* ILFD $X \rightarrow Y$ if for every possible tuple $r \in R$, such that X holds, it is also true that Y holds in r. We say that a relation R *violates* ILFD $X \rightarrow Y$ iff R does not satisfy the ILFD.

Although ILFDs can be modeled using propositional logic, it can also be modeled in first order logic as program clauses [9]. ILFDs are defined for a specific relation modeling an entity set. Their first order logic representation always involves only one predicate symbol. In this case, representing ILFDs using propositional logic can make the ILFD reasoning process simpler.

5.1. *COMPARING ILFDS WITH FDS*

In Section 4.1, we describe the similarities and differences between ILFDs and FDs. The relationship between FDs and ILFDs can be seen from the following proposition:

PROPOSITION 2. *If for each combination of values $a_1, \ldots, a_m$ in the domains of $A_1, \ldots, A_m$, respectively, there is an ILFD $((A_1 = a_1) \wedge \cdots \wedge (A_m = a_m)) \rightarrow ((B_1 = b_1) \wedge \cdots \wedge (B_n = b_n))$ that holds in the relation R, then the FD$\{A_1, \ldots, A_m\} \rightarrow \{B_1, \ldots, B_n\}$ also holds in R.*

Proof. Suppose we are given all ILFDs of the form $((A_1 = a_1) \wedge \cdots \wedge (A_m = a_m)) \rightarrow ((B_1 = b_1) \wedge \cdots \wedge (B_n = b_n))$, such that each ILFD corresponds to a combination of $A_1, \ldots, A_m$ values. Now given any two tuples, t_1 and t_2 from R, if t_1 and t_2 agree in their $A_1, \ldots, A_m$ values, by the appropriate ILFD, we can infer that t_1 and t_2 also agree in their $B_1, \ldots, B_n$. Therefore, the FD $\{A_1, \ldots, A_m\} \rightarrow (B_1, \ldots, B_n\}$ holds in R. □

Notice that the converse of the above theorem is not necessarily true because FDs do not suggest particular values for the attributes involved. Based on the similarity between FDs and ILFDs, we develop an ILFD theory analogous to that of FDs.

5.2. *ARMSTRONG'S AXIOMS FOR IFLDS*

Let F be a set of ILFDs for relation scheme R modeling a set of related real-world entities and let $P \rightarrow Q$ be an ILFD. $P \rightarrow Q$ is said to be inferred from F, i.e., $F \models P \rightarrow Q$, iff every tuple r in R that satisfies the ILFDs in F also satisfies $P \rightarrow Q$.

DEFINITION. The *closure* of a set of ILFDs F, denoted by F^+, is the set of ILFDs that are logically implied by F; i.e.,

$$F^+ = \{P \rightarrow Q | F \models (P \rightarrow Q)\}.$$

As in the case of FDs, the set of inference rules for ILFDs can be defined. Because this set of rules is similar to Armstrong's axioms for FDs, we call them Armstrong's axioms for ILFDs.

1. *Reflexivity.* $F \models ((X_1 \wedge \cdots \wedge X_m) \rightarrow (X_1 \wedge \cdots \wedge X_n))$, where $m \leqslant n$. ILFDs of this form are known as *trivial ILFDs* because they hold in any entity set and do not depend on F.

2. *Augmentation.* Let X, Y, and Z be conjunction of propositional symbols. Now, if $F \models (X \rightarrow Y)$, then $F \models ((X \wedge Z) \rightarrow (Y \wedge Z))$.

3. *Transitivity.* Let X, Y, and Z be conjunction of propositional symbols. Now, if $F \models (X \rightarrow Y)$ and $F \models (Y \rightarrow Z)$, then $F \models (X \rightarrow Z)$.

LEMMA 1. *Armstrong's axioms for ILFDs are sound.*

Proof. To prove that Armstrong's axioms for ILFDs are sound, we need to prove that if $X \rightarrow Y$ is deduced from F using the axioms, then $X \rightarrow Y$ is true in any relation in which the ILFDs of F are true.

It is obvious that the *reflexivity axiom* is sound because we cannot have a relation with a tuple in which X holds but some subset of X does not hold.

To prove that the *augmentation axiom* is sound, suppose we have a relation r that satisfies $X \rightarrow Y$, and yet there is a tuple α in r such that $X \wedge Z$ holds in α but $Y \wedge Z$ does not. Because both Z and X must hold in α, it must be the case that Y does not. However, this contradicts the initial assumption of $X \rightarrow Y$. Therefore, the augmentation axiom is sound.

To prove that the *transitivity axiom* is sound, suppose we have a relation r that satisfies $X \rightarrow Y$ and $Y \rightarrow Z$, but there is a tuple α in r such that X holds but Z does not hold. Because Z does not hold for α, neither must Y. Similarly, we conclude that X does not hold for α. However, this contradicts the initial assumption that X holds for α. Therefore, the transitivity axiom is sound. □

EXAMPLE

Let $R(A,B,C)$ be a relation scheme and $F = \{(A = a1) \rightarrow (B = b1), (B = b1) \rightarrow (C = c1)\}$. Let P, Q, and R denote $A = a1$, $B = b1$, and $C = c1$, respectively. Now, $F^+ = \{P \rightarrow P,\ Q \rightarrow Q,\ R \rightarrow R,\ (P \wedge Q) \rightarrow P,\ (P \wedge Q) \rightarrow Q,\ (P \wedge R) \rightarrow P,\ (P \wedge R) \rightarrow R,\ (Q \wedge R) \rightarrow Q,\ (Q \wedge R) \rightarrow R,\ (P \wedge Q \wedge R) \rightarrow P,\ (P \wedge Q \wedge R) \rightarrow Q,\ (P \wedge Q \wedge R) \rightarrow R, \ldots\}$.

Notice that from the above three basic axioms, several other inference rules on ILFDs can be derived, as shown below.

LEMMA 2.

1. *Union Rule.* $\{X \rightarrow Y,\ X \rightarrow Z\} \models X \rightarrow (Y \wedge Z)$.
2. *Pseudotransitivity Rule.* $\{X \rightarrow Y,\ (W \wedge Y) \rightarrow Z\} \models (W \wedge X) \rightarrow Z$.
3. *Decomposition Rule. If* $X \rightarrow (Y \wedge Z)$ *holds, then* $X \rightarrow Z$ *holds.*

The proof is similar to that in FDs.

DEFINITION (Closure of a set of proposition symbols). Let F be a set of IFLDs and let X be a set of proposition symbols. The *closure of* X with respect to F, denoted by X_F^+, is the set of proposition symbols A such that $X \rightarrow A$ can be deduced from F by Armstrong's axioms for ILFDs.

Having defined the above terms, we are now ready to prove the important theorem about Armstrong's axioms for ILFDs.

THEOREM 1. *Armstrong's axioms are sound and complete.*

Proof. Armstrong's axioms are sound from Lemma 1. Therefore, only completeness is left to be proved. Let F be a set of ILFDs and suppose $X \rightarrow Y$ cannot be inferred from the axioms, but is satisfied by all instances of relation R. This means that Y is not in X_F^+. Let r be an instance of R such that all propositional symbols in X_F^+ are true, but all other propositional symbols are false. This relation r certainly satisfies the axioms. For $X \rightarrow Y$ to hold, Y must be true in R, i.e., $Y \subseteq X^+$. This contradicts the fact that $X \rightarrow Y$ cannot be inferred from the axioms. Therefore, Armstrong's axioms are sound and complete. □

Similar to the closure of a set of FDs, the closure of a set of ILFDs is expensive to compute. This is because the closure of a set of ILFDs may contain a huge number of ILFDs that can be inferred from the original ILFD set.

On the other hand, computing the closure X_F^+ of a set of propositional symbols X with respect to a set of ILFDs F is relatively easier. Essentially, the algorithm for computing X_F^+ is the same as that for computing the closure of a set of attributes with respect to a set of FDs.

6. IMPLEMENTATION OF ENTITY-IDENTIFICATION TECHNIQUE USING ILFDS

In this section, we describe a Prolog implementation of our proposed entity-identification technique.[7] Prolog was chosen because its basic constructs allow us to easily represent both the ILFDs and the source relations. Furthermore, its well-known operational semantics guarantees that the result obtained for a Prolog program is always a logical consequence of the axioms contained in the program.

Our implementation stores the source relations and ILFDs as facts and rules, respectively. By allowing the user to specify the *extended key*, our system dynamically creates the rule that defines the *matching table*. Using the matching table, the integrated relation can be constructed. Every time an extended key is specified by the user, our system verifies that it does not violate the *soundness criteria* of the matching result. That is, there should not be a tuple from one source relation being matched to more than one tuple from the other source relation.

In the following, we describe our systems using Example 3 of Section 4.2. A complete listing of the Prolog program is given in the Appendix.

6.1. REPRESENTATION OF RELATIONS AND ILFDS

Instead of storing each tuple in the source relation as a Prolog fact, we assign a unique id to the tuple and store it as multiple facts. Each of these facts contain a binary predicate that relates the tuple id to an attribute. By representing a tuple as facts with binary predicates, we provide the flexibility of augmenting the tuple with additional attributes. Moreover, the representation of ILFDs becomes straightforward. For example, the first tuple in relation R,

$$(\text{“TwinCities”}, \text{“Chinese”}, \text{“Co.B2”}),$$

has been represented as

$$r\_rname(r1, twincities),$$

$$r\_cui(r1, chinese),$$

$$r\_str(r1, co\_B2).$$

[7] The Prolog interpreter we used is the SB-Prolog System, Version 3.0 [6].

The ILFD

$$(\textit{Restaurant.speciality} = \text{"Hunan"}) \rightarrow (\textit{Restaurant.cuisine} = \text{"Chinese"})$$

is represented as the Prolog rule

$$s\_cui(Rid, chinese) :- s\_spec(Sid, hunan), !.$$

A cut (!) is given at the end of an ILFD to prevent other ILFDs from being used once the former ILFD has successfully derived the attribute value. Note that the tuple id, other than being used for relating the attributes belonging to the same tuple in a source relation, does not participate in the actual entity-identification process because ILFDs and extended key do not use the tuple id as an identifying attribute.

6.2. REPRESENTATION AND REASONING WITH MISSING INFORMATION

Our system uses NULL values to represent missing information in both the matching table and the integrated table. In the entity-identification process, tuples from the source relations may have some missing extended key attributes. For example, the *speciality* attribute of the entity modeled by the R-tuple ("TwinCities","Indian","Co.B3") cannot be derived by any given ILFD. Missing information must also be captured in the integrated table because source relations may model different sets of entities. If a tuple from a source relation does not match any tuple from the other source relation, the missing attributes are represented as NULLs. In our prototype system, NULL values are assigned as the default values for those attributes whose values are neither given as facts nor derivable from the ILFDs. Because the Prolog interpreter searches for usable rules in a top down manner, we implemented the default NULL values by asserting them only after all ILFDs have failed to assign the non-NULL values. For example, the following two assertions can be found below all the source relation assertions and ILFDs:

$$r\_spec(Rid, null),\ s\_cui(Sid, null).$$

Because a NULL value is represented as an ordinary symbol in our system, extra care must be taken when NULL values are involved in equality tests, as in the rule defining the matching table. In this case, we do not want a

NULL value to be equated with another NULL value. Hence, we implemented a *non_null_eq* predicate that only holds for comparison between non-NULL values.

6.3. DESCRIPTION OF ENTITY-IDENTIFICATION PROCESS

Having specified the source relations as Prolog facts and the ILFDs as rules, our system allows the users to specify an extended key dynamically and generates the rule defining the matching table.

To specify an extended key, the command **setup_extkey** can be invoked. The **setup_extkey** invocation lists the candidate attributes for the extended key and asks the user to select some of all of them as the extended key. Candidate attributes are those that are common among the source relations and have been asserted to be semantically equivalent. In our system, we assume that such information has been supplied a priori. The following shows how the **setup_extkey** is invoked. To improve the readability, all user responses have been written in italics.

```
| ?- setup_extkey.
      [0] Name: (r_name,s_name)
      [1] Spec: (r_spec,s_spec)
      [2] Cui: (r_cui,s_cui)
Please input the no. of keys: 3
Please input the keys:
      key 1=0
      key 2=1
      key 3=2
The new definition for the matching table :
      matchtable(R_name,R_cui,S_name,S_spec) :-
               r_name(R,R_name),s_name(S,S_name),
               r_spec(R,R_spec),s_spec(S,S_spec),
               r_cui(R,R_cui),s_cui(S,S_cui),
               non_null_eq(R_name,S_name),
               non_null_eq(R_spec,S_spec),
               non_null_eq(R_cui,S_cui).
Message: The extended key is verified.
yes
```

Once the extended key is selected, a new rule defining the matching table is created. Due to the difficulty of constructing rules using Prolog, a small C program (getkey) is written to create the rule defining the matching table. Each time a new extended key is specified, our system displays the rule defining the matching table. It also verifies that no tuple from a source relation is matched with more than one tuple from another relation in the new matching table. If an anomaly occurs, a warning message is displayed.

```
| ?- setup_extkey.
      [0] Name: (r_name,s_name)
      [1] Spec: (r_spec,s_spec)
      [2] Cui: (r_cui,s_cui)
Please input the no. of keys: 1
Please input the keys:
      key 1=0
The new definition for the matching table :
    matchtable(R_name,R_cui,S_name,S_spec) :-
              rname(R,R_),s-name(S,S_name),
              r_spec(R,R_spec),s_spec(S,S_spec),
              r_cui(R,R_cui),s_cui(S,S_cui),
              non_null_eq(R_name,S_name).
Message: The extended key causes unsound matching
result.
yes
```

With the availability of a rule defining the matching table, the user can invoke **print_matchable** and **print_integ_table** to display the matching table and the integrated table respectively. For example, if we have selected {*Name*, *Spec*, *Cui*} as our extended key, the display of matching table and integration table is

```
| ?- print_matchtable.


                   matching table

      r_name       r_cui     s_name      s_spec

      anjuman      indian    anjuman     mughalai
      itsgreek     greek     itsgreek    gyros
      twincities   chinese   twincities  hunan
      yes


| ?- print_integ_table.
```

integrated table

r_name	r_cui	r_spec	s_name	s_cui	s_spec	r_str	s_cty
anjuman	indian	mughalai	anjuman	indian	mughalai	le_salle_ave	minneapolis
itsgreek	greek	gyros	itsgreek	greek	gyros	front_ave	ramsey
null	null	null	twincities	chinese	sichuan	null	hennepin
twincities	chinese	hunan	twincities	chinese	hunan	co_B2	roseville
twincities	indian	null	null	null	null	co_B3	null
villagewok	chinese	null	null	null	null	wash.ave	null
yes							

7. CONCLUSION

In this paper, we address the entity-identification problem, an instance level problem, in integrating pre-existing autonomous databases. We formulate entity identification as a matching problem and identify desirable properties, i.e., soundness and completeness, for an ideal entity-identification technique. In order to integrate instances from several autonomous databases, most existing approaches assume that the original relations have at least one common candidate key, and key equivalence is a valid identity rule. By using the extended key equivalence identity rule and semantic information such as instance level functional dependencies (ILFD), we are able to relax these restrictions and allow relations with no common candidate key to be integrated. Such semantic information can be supplied either by database administrators during schema integration or through some knowledge acquisition tools. The applicability of our approach has been demonstrated using an example. We have also described an implementation of our entity-identification technique using Prolog.

Entity identification is a major task to be dealt with in database integration. In processing a federated database query, entity identification has to be performed whenever the information about real-world entities exists in different databases. Our ongoing research is developing mechanisms to do so.

APPENDIX A: A PROLOG IMPLEMENTATION OF THE PROPOSED ENTITY-IDENTIFICATION TECHNIQUE

The following is a listing of the Prolog program that implements our entity identification technique.

```
/*
        Entity Identification Example -- (Restaurant)

*/

/*      Table R (name, cuisine, street)

*/

r_name(r1,twincities).
r_cui(r1,chinese).
r_str(r1,co_B2).
```

```
r_name(r2,twincities).
r_cui(r2,indian).
r_str(r2,co_B3).

r_name(r3,itsgreek).
r_cui(r3,greek).
r_str(r3,front_ave).

r_name(r4,anjuman).
r_cui(r4,indian).
r_str(r4,le_salle_ave).

r_name(r5,villagewok).
r_cui(r5,chinese).
r_str(r5,wash_ave).

/*      Table S (name, speciality, county)

*/

s_name(s1,twincities).
s_spec(s1,hunan).
s_cty(s1,roseville).

s_name(s2,twincities).
s_spec(s2,sichuan).
s_cty(s2,hennepin).

s_name(s3,itsgreek).
s_spec(s3,gyros).
s_cty(s3,ramsey).

s_name(s4,anjuman).
s_spec(s4,mughalai).
s_cty(s4,minneapolis).

/*      ILFDs

*/

s_cui(Sid,chinese) :- s_spec(Sid,hunan),!.
s_cui(Sid,chinese) :- s_spec(Sid,sichuan),!.
s_cui(Sid,greek) :- s_spec(Sid,gyros),!.
s_cui(Sid,indian) :- s_spec(Sid,mughalai),!.

r_spec(Rid,hunan) :-
r_name(Rid,twincities),r_str(Rid,co_B2),!.
r_spec(Rid,mughalai) :-
r_name(Rid,anjuman),r_str(Rid,le_salle_ave),!.

r_cty(Rid,ramsey) :- r_str(Rid,front_ave),!.

r_spec(Rid,gyros) :-
r_name(Rid,itsgreek),r_cty(Rid,ramsey),!.

r_spec(Rid,null).
s_cui(Sid,null).

/* Extended Relations */

rr(Name,Cui,Spec,Str) :- r_(Rid,Name),r_cui(Rid,Cui),
                        r_str(Rid,Str),
r_spec(Rid,Spec).
ss(Name,Cui,Spec,Cty) :- s_(Sid,Name),
                        s_spec(Sid,Spec),
                        s_cty(Sid,Cty),
                        s_cui(Sid,Cui).

/* Integrated Relation */

/* oldrs is more general since it does not concern
abt resolution between common attributes */

rs(RName,RCui,RSpec,SName,SCui,SSpec,RStr,SCty) :-
        matchtable(RName,RCui,SName,SSpec),
        rr(RName,RCui,RSpec,RStr),
        ss(SName,SCui,SSpec,SCty).
rs(RName,RCui,RSpec,null,null,null,RStr,null) :-
        rr(RName,RCui,RSpec,RStr),
        not matchtable(RName,RCui,_,_).
rs(null,null,null,SName,SCui,SSpec,null,SCty) :-
        ss(SName,SCui,SSpec,SCty),
        not matchtable(_,_,SName,SSpec).
```

```
/* Verification of Extended Key */

length([],0).
length([X | Xs],N+1) :- length(Xs,N).

if_then_else(P,Q,R) :- P,!,Q.
if_then_else(P,Q,R) :- R.

non_null_eq(A,B) :- non A-null, not B-null, A=B.

matched_R_keys(A,B) :- matchtable(A,B,C,D).
matched_S_keys(C,D) :- matchtable(A,B,C,D).

correct :- bagof([A,B],matched_R_keys(A,B),M1),
        setof([C,D],matched_R_keys(C,D),M2),
        bagof([E,F],matched_S_keys(E,F),M3),
        setof([G,H],matched_S_keys(G,H),M4),
        length(M1,N1),length(M2,N2),length(M3,N3),
        length(M4,N4),
        N1=N2, N3=N4.

acknowledge :- name(X, ''Message: The extended key is
verified.''),print(X),nl.
warning :- name(X, ''Message: The extended key causes
unsound matching result.''),print(X),nl.

verify :- if_then_else(correct,acknowledge,warning).

/* Setup Extended key */

setup_extkey :- not(system('getkey')),
        abolish(matchtable,4),
        consult(extkeyeq),verify.

/* Print the Extended R & S Tables */
print_RRtable :- setof([A,B,C,D],rr(A,B,C,D),RRRecs),
        nl,prtRRhdg,printreclist(RRRecs).

print_SStable :- setof([A,B,C,D],ss(A,B,C,D),SSRecs),
        nl,prtSShdg,printreclist(SSRecs).
```

```
/* Print the Matching Table */
print_matchtable :-
setof([A,B,C,D],matchtable(A,B,C,D),MatchRecs),
        nl,prtmatchtabhdg,printreclist(MatchRecs).

/* Print the Integrated Table */
print_integ_table :-
        setof([A,B,C,D,E,F,G,H],rs(A,B,C,D,E,F,G,H),
        ntegRecs),
        nl,prtintegtabhdg,printreclist(IntegRecs).

/* Print Utility */

prtRRhdg :- name(X,''extended R table''),
                print_ar(30,X),nl,
                print_ar(30,---------------),nl,
                print_al(15,r_name),
                print_al(15,r_cui),
                print_al(15,r_spec),
                print_al(15,r_str),nl,
                name(Z,''------- ''),print_al(15,Z),
                print_al(15,Z),
                print_al(15,Z),print_al(15,Z),nl.

prtSShdg :- name(X,''extended S table''),
                print_ar(30,X),nl,
                print_ar(30,---------------),nl,
                print_al(15,s_name),
                print_al(15,s_cui),
                print_al(15,s_spec),
                print_al(15,s_cty),nl,
                name(Z,''------- ''),print_al(15,Z),
                print_al(15,Z),
                print_al(15,Z),print_al(15,Z),nl.

prtmatchtabhdg :- name(X,''matching table''),
                print_ar(30,X),nl,
                print_ar(30,---------------),nl,
                print_al(15,r_name),
                print_al(15,r_cui),
                print_al(15,s_name),
                print_al(15,s_spec),nl,
                name(Z,''------- ''),print_al(15,Z),
                    print_al(15,Z),
                    print_al(15,Z),print_al(15,Z),nl.

prtintegtabhdg :- name(X, ''integrated table ''),
                    print_ar(30,X),nl,
                    print_ar(30,----------------),nl,
                    print_al(15,r_name),
                    print_al(15,r_cui),
                    print_al(15,r_spec),
                    print_al(15,s_name),
                    print_al(15,s_cui),
                    print_al(15,s_spec),
                    print_al(15,r_str),
                    print_al(15,s_cty),nl,
                    name(Z, ''------- ''),
                    print_al(15,Z),print_al(15,Z),
                    print_al(15,Z),
                    print_al(15,Z),print_al(15,Z),
                    print_al(15,Z),
                    print_al(15,Z),print_al(15,Z),nl.

printrec([]) :- nl.
printrec([A | Alist]) :-
print_al(15,A),printrec(Alist).

printreclist([]) :- nl.
printreclist([X | Xlist]) :-
printrec(X),printreclist(Xlist).
```

REFERENCES

1. R. Ahmed, P. DeSmedt, W. Du, B. Kent, M. Ketabchi, W. Litwin, A. Rafii, and M-C. Shan. The pegasus heterogeneous multidatabase system. *IEEE Computer* 24:19–27 (1991).
2. E. Bertino. Integration of heterogeneous data repositories by using object-oriented views. In *Proceedings of the 1st International Workshop on Interoperability in Multidatabase Systems*, 1991.
3. M. Batini, C. Lenzirini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surveys* 18:323–364 (1986).
4. A. Chatterjee and A. Segev. Data manipulation in heterogeneous databases. *SIGMOD Record* 20:64–68 (1991).
5. U. Dayal. Processing queries over generalized hierarchies in a multidatabase systems. In *Proceedings of the 9th VLDB Conference*, 1983.

6. S. K. Debray. The SB-Prolog System, Version 3.0, 1988.
7. L. G. DeMichiel. Resolving database incompatibility: An approach to performing relational operations over mismatched domains. *IEEE Trans. Knowledge Data Eng.* 1:485–493 (1989).
8. W. Kent. The breakdown of the information model in mdbss. *SIGMOD Record* 20:10–15 (1991).
9. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1984.
10. J. A. Larson, S. B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Trans. Software Eng.* 15:449–463 (1989).
11. D. V. McDermott. Non-monotonic logic 1. *Artificial Intelligence* 13:41–72 (1980).
12. E. J. Neuhold, W. Kent, and M-C. Shan. Object identification in interoperable database systems. In *Proceedings of the 1st Workshop on Interoperability in Multidatabase Systems*, 1991.
13. C. Pu. Key equivalence in heterogeneous databases. In *Proceedings of the 1st International Workshop on Interoperability in Multidatabase Systems*, 1991.
14. A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys* 22:183–236 (1990).
15. F. S-C. Tseng, A. L. P. Chen, and W-P. Yang. A probabilistic approach to query processing in heterogeneous database systems. In *RIDE TQP 92*, 1992.
16. J. Ullman. *Principles of Database Systems*. Computer Science Press, Rockville, MD, 1982.
17. S. Widjojo, R. Hull, and D. Wile. A specification approach to merging persistent object bases. In *Workshop on Persistent Object Bases*, 1990.
18. Y. R. Wang and S. E. Madnick. The inter-database instance identification problem in integrating autonomous systems. In *Proceedings of the 5th International Conference on Data Engineering*, 1989.
19. C. Yu, W. Sun, S. Dao, and D. Kersey. Determining relationships among attributes for interoperability of multi-database systems. In *Proceedings of the 1st International Workshop on Interoperability in Multidatabase Systems*, 1991.

Received 12 January 1994; revised 23 March 1995