An ontology based approach to the integration of entity-relationship schemas

Qi He * Tok Wang Ling

Dept. of Computer Science, School of Computing, National University of Singapore, 3 Science Drive 2, Singapore 117543

Abstract

In schema integration, schematic discrepancies occur when data in one database correspond to metadata in another. We explicitly declare the context that is the meta information relating to the source, classification, property etc of entities, relationships or attribute values in entity-relationship (ER) schemas. We present algorithms to resolve schematic discrepancies by transforming metadata into the attribute values of entity types, keeping the information and constraints of original schemas. Although focusing on the resolution of schematic discrepancies, our technique works seamlessly with the existing techniques resolving other semantic heterogeneities in schema integration.

Key words: schematic discrepancy, schema integration, entity-relationship approach, ontology

1 Introduction

Schema integration involves merging several schemas into an integrated schema. More precisely, [4] defines schema integration as "the activity of integrating the schemas of existing or proposed databases into a global, unified schema". It is regarded as an important work to build a heterogeneous database system [5] [21] (also called *multidatabase system* or *federated database system*), to integrate data in a data warehouse, or to integrate user views in database design. In schema integration, people have identified different kinds of semantic heterogeneities among component schemas: naming conflict (homonyms and

* Corresponding author.
Email addresses: heqi@comp.nus.edu.sg (Qi He), lingtw@comp.nus.edu.sg (Tok Wang Ling).

synonyms), key conflict, structural conflict [3] [15], and constraint conflict [14] [19].

A less touched problem is schematic discrepancy, i.e., the same information is modelled as data in one database, but metadata in another. The following example illustrates schematic discrepancy in ER schemas. To focus our contribution and simplify the presentation, in the example below, schematic discrepancy is the only kind of conflicts among schemas.

Example 1 *Suppose we want to integrate the supply information of products from three databases DB*1*, DB*2 *and DB*3 *(Figure 1). These databases record similar information, i.e., product numbers, product names, suppliers and the supplying prices in each month. In DB*1*, the supply relationships are modelled as a ternary relationship type SUP. In DB*2*, the entity type JAN_PROD models the products supplied in the month of January, and the attributes S1_PRICE, . . . , Sn_PRICE means the prices of the products by the suppliers S1, . . . , Sn. For example, the attribute S1_PRICE of the entity type JAN_PROD means the prices of the products supplied in January by the supplier S1. In DB*3*, the relationship type JAN_SUP models the supply relationships between products and suppliers in January. Note that JAN_SUP is a selection of the ternary relationship type SUP of DB*1 *(when the value of M# is 'JAN').*

In relational databases, these ER schemas correspond to the following relational schemas (i.e., each entity type having more than one attribute and each relationship type would be transformed into a relation):

DB1: PROD(P#, PNAME), SUP(P#, S#, M#, PRICE)

DB2: JAN_PROD(P#, PNAME, S1_PRICE, . . . , Sn_PRICE),
⋮
DEC_PROD(P#, PNAME, S1_PRICE, . . . , Sn_PRICE)

DB3: PROD(P#, PNAME),
JAN_SUP(P#, S#, PRICE),
⋮
DEC_SUP(P#, S#, PRICE)

The schemas of Figure 1 are schematically discrepant from each other. For example, the values of the attribute M# in DB1 correspond to the metadata of the relationship types (i.e., the names of the relationship types) in DB3. The values of the attribute M# in DB1 correspond to the metadata of the entity types in DB2, and the values of the attribute S# in DB1 correspond to the metadata of the attributes S1_PRICE, . . . , Sn_PRICE in DB2.

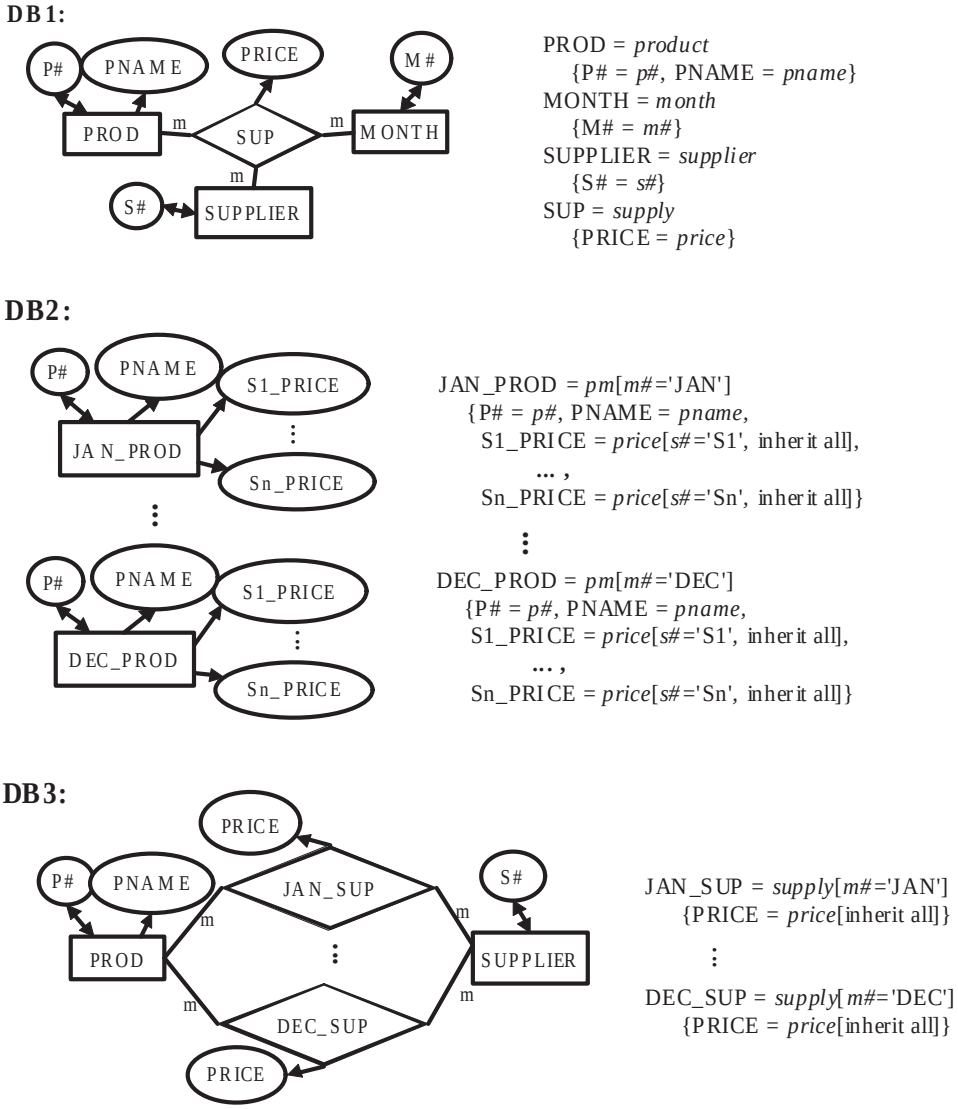


Fig. 1. Schematic discrepancies: months and suppliers modelled differently as the attribute values or meta information in $DB1$, $DB2$ and $DB3$

In Section 4, we will resolve schematic discrepancies by transforming metadata into attribute values, e.g., transforming DB2 and DB3 into a form of DB1, and then merge the transformed schemas. The statements on the right side of Figure 1 specify the meta information of the schemas using a shared ontology, which will be explained in Section 3. □

Schematic discrepancy arises frequently since the names of schema constructs often capture some intuitive semantic information. Real examples of such disparity abound [11] [12] [18]. Originally raised as a conflict to be resolved in schema integration, schematically discrepant structures have been used to solve some interesting problems:

- In [18], Miller identified three scenarios in which schematic discrepancies may occur, i.e., database integration, data publication on the web and physical data independence.
- In e-commerce, Agrawal et al [2] argued that the new generation of e-commerce applications require the data schemas that are constantly evolving and sparsely populated. They believed that a vertical representation of objects (in which attribute names are modelled as data values) is much better on storage and querying performance than the conventional horizontal row representation. On the other hand, to facilitate writing queries, they need to create the horizontal views of vertical tables.
- In data warehousing, users usually require generating two-dimensional report tables which are schematically discrepant from fact data.

We adopt a semantic approach to resolve schematic discrepancies in the integration of ER schemas. One of the outstanding features of our proposal is that we preserve cardinality constraints in the transformation/integration of ER schemas. Cardinality constraints, in particular, functional dependencies (FDs) and multivalued dependencies (MVDs), are useful in verifying lossless schema transformation [8], schema normalization and semantic query optimization [9] [19] in multidatabase systems.

The rest of the paper is organized as follows. Section 2 comprises an introduction to the ER approach and an ontology-based approach to the integration of ER schemas. Section 3 to 5 are the main contributions of this paper. In Section 3, we first introduce the concepts of ontology and context which are used to specify the meta information of ER schemas. In Section 4, we present algorithms to resolve different kinds of schematic discrepancies in schema integration. In Section 5, we show that our resolution algorithms preserve information and cardinality constraints in schema transformation. In Section 6, we compare our work with related work. Section 7 concludes the whole paper.

2 Preliminaries

2.1 ER Approach

In the ER model, an entity is an object in the real world and can be distinctly identified. An *entity type* is a collection of the similar entities that have the same set of predefined common attributes. Attributes can be *single-valued*, i.e., 1:1 (one-to-one) or m:1 (many-to-one), or *multivalued*, i.e., 1:m (one-to-many) or m:m (many-to-many). A minimal set of attributes of an entity type E which uniquely identifies E is called a *key* of E. An entity type may have more than one key and we designate one of them as the *identifier* of the entity type.

A relationship is an association among two or more entities. A *relationship type* is a collection of the similar relationships that satisfy a set of predefined common attributes. A minimal set of attributes (including the identifiers of participating entity types) in a relationship type R that uniquely identifies R is called a *key* of R. A relationship set may have more than one key and we designate one of them as the *identifier* of the relationship type.

The cardinality constraints of ER schemas incorporate FDs and MVDs. For example, in the ER schema of Figure 2, let K1, K2 and K3 be the identifiers of the entity types E1, E2 and E3, we have:

K1 $\rightarrow$ A1 and A1 $\rightarrow$ K1, as A1 is a one-to-one attribute of E1;
K2 $\rightarrow$ A2, as A2 is a many-to-one attribute of E2;
K3 $\twoheadrightarrow$ A3, as A3 is a many-to-many attribute of E3;
K1, K2 $\rightarrow$ K3, as {K1, K2} is the identifier of the relationship type R, and the cardinality of E3 is 1 in R;
K1, K2 $\rightarrow$ B, as B is a many-to-one attribute of R.

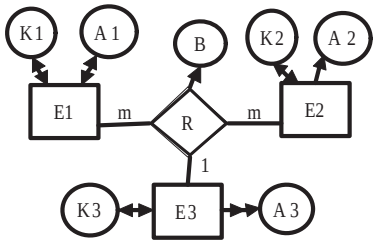


Fig. 2. Dependencies in ER schema

2.2 Ontology-based Schema Integration

An ontology is a description of the concepts in the universe of discourse for the purpose of enabling knowledge sharing and reuse. It provides a common type system for information exchange between disparate systems. Various approaches have been introduced to build the ontology for a federation of information sources. In this paper, we'll use domain-specific ontologies to represent the meta information of ER schemas.

In [16], Ling et al proposed an ER-based federated database system where local schemas modelled in the relational, object-relational, network or hierarchical models are first translated into corresponding ER export schemas before they are integrated. Database reverse engineering which constructs ER conceptual schemas from existing relational, object-relational, etc databases facilitate the integration of these databases. Our approach is an extension to theirs by using an ontology to provide the semantics necessary for schema integration. In our proposal, source data could be in different data models, e.g., relational model,

object relational model, etc. We first translate them to ER schemas the meta information of which are specified using a shared ontology. The cardinality constraints of the ER schemas are specified from the integrity constraints of source databases. Then we integrate the ER schemas, in which semantic heterogeneities among the schemas are resolved. The integrity constraints on the integrated schema are derived from the constraints on the component schemas at the same time.

In such a framework, the transformations from source schemas to ER schemas can be done semi-automatically (we have developed a semi-automatic tool to help users transform schemas). Then the detection and reconciliation of semantic heterogeneities are done automatically by systems (in this paper, we will study some key techniques to resolve schematic discrepancies in such integration systems). Such an ontology-based framework scale-up efficiently given the complexity involved in integrating a large number of schemas, compared with the assertion-based approach in which integration assertions are used to relate equivalent constructs in schemas [15] [22].

3 Ontology and Context

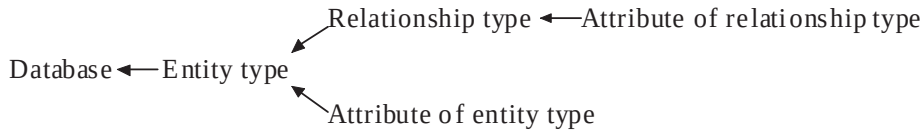
We treat an ontology as the specification of the representational vocabulary for a shared domain of discourse which includes the definitions of entity types, relationship types, attributes of entity types and attributes of relationship types. We present ontologies at a conceptual level, which could be implemented by ontology languages.

For example, suppose an ontology *SupOnto* describes the concepts in the universe of product supply. It includes entity types *product*, *month*, *supplier*, a ternary relationship type *supply* among *product*, *month* and *supplier*, a binary relationship type *pm* that is a projection of the relationship type *supply* onto the entity types *product* and *month*, attributes *p#*, *pname*, *s#*, *m#* (the values of *m#* are 'JAN', 'FEB', ..., 'DEC') and *price* that is an attribute of *supply*. Note we use lower case italic words to represent the types of an ontology, in contrast to capitals for the schema constructs of an ER schema.

Conceptual modelling is always done within a particular context. In particular, the context of an entity type, relationship type or attribute is the meta-information relating to its source, classification, property etc.

For example, in $DB2$ of Figure 1, the entity type JAN_PROD has a context of the month January (i.e., 'JAN') to indicate the product entities in the type are supplied in January.

In an ER schema, contexts are usually at four levels: *database, entity type, relationship type* and *attribute*. In general, we have the following hierarchy of inheritance relations between contexts at different levels:



That is, an entity type may "inherit" a context from a database (i.e., the context of a database applies to the entities), a relationship type may "inherit" a context from its involving entity types and so on.

For example, in $DB2$ of Figure 1, the attribute S1_PRICE of the entity type JAN_PROD inherits the context of the month January from its owner entity type. In other words, the prices of products supplied by the supplier S1 are dependent on months.

The inheritance hierarchy actually reflects the order in which an ER schema is built up. Given an application to be modelled, we first identify entity types, then the attributes of the entity types and the relationship types among the entity types, and finally the attributes of the relationship types. Correspondingly, we first decide the context of a database in which an ER schema is modelled, then the contexts of entity types which may inherit a context from the database, and so on.

We will give a formal representation of context below. Note as the context of a database would be handled in the entity types which inherit it, we ignore it in the following definition.

Definition 1 *Given an ontology, we represent an entity type (a relationship type, or an attribute) E of an ER schema as:*

$$E = T[C_1 = c_1, \ldots, C_m = c_m, inherit\ C_{m+1}, \ldots, C_n]$$

where T is a type of the ontology (T is an entity type or relationship type if E is an entity type, is a relationship type if E is a relationship type, or is an attribute if E is an attribute), $C_1, \ldots, C_n$ are attributes of the ontology, and each c_i is a value of C_i for each $i = 1, \ldots, m$. $C_{m+1}, \ldots, C_n$ respectively have a value of $c_{m+1}, \ldots, c_n$ which are stated in a higher level context (i.e. the context of a database if E is an entity type, the contexts of entity types if E is a relationship type, or the context of an entity type/relationship type if E is an attribute).

This representation means that each instance of E is an instance of T, and satisfies the conditions $C_i = c_i$ for each $i = 1, \ldots, n$. $C_1, \ldots, C_n$ with the

values constitute the ***context*** *within which E is defined; we call them* ***meta-attributes****, and their values* ***metadata*** *of E. We say E* ***inherits*** *the context $\{C_{m+1} = c_{m+1}, \ldots, C_n = c_n\}$. If E inherits all the meta-attributes with the values of a higher level context, we simply represent it as:*

$$E = T[C_1 = c_1, \ldots, C_m = c_m, inherit\ all]$$

For easy reference, we call the set $\{C_1 = c_1, \ldots, C_m = c_m\}$ the ***self context****, and $\{C_{m+1} = c_{m+1}, \ldots, C_n = c_n\}$ the* ***inherited context*** *of E.* □

Either self or inherited contexts could be empty. In the example below, we represent the entity types, relationship types and attributes in Figure 1 using the ontology *SupOnto*.

Example 2 *In Figure 1, the entity type JAN_PROD of DB2 is represented as:*

$$JAN\_PROD = pm[m\# = \text{`JAN'}].$$

The context of JAN_PROD is m# ='JAN'. This means that JAN_PROD corresponds to a relationship type pm when the month is January, i.e., the products supplied in January.

Also in DB2, the attribute S1_PRICE of the entity type JAN_PROD is represented as:

$$S1\_PRICE = price[s\# = \text{`S1'}, inherit\ all].$$

The self context of S1_PRICE is s#='S1', and the inherited context (from the entity type JAN_PROD) is m#='JAN'. This means that each value of S1_PRICE of the entity type JAN_PROD is a price of a product supplied by supplier S1 in January.

In DB3, the relationship type JAN_SUP is represented as:

$$JAN\_SUP = supply[m\# = \text{`JAN'}].$$

This means that each relationship of JAN_SUP corresponds to a relationship of supply when the month is January.

Also in DB3, the attribute PRICE of the relationship type JAN_SUP is represented as:

$$PRICE = price[inherit\ all].$$

PRICE inherits the context m#='JAN' from its relationship type JAN_SUP. This means that each value of PRICE is a supplying price in January. □

In schema integration, contexts should be declared by the owners of source schemas. Once declared, our integration system can detect the schema matching information from the contexts automatically. For example, two entity types, two relationship types or two attributes are ***equivalent*** if they correspond to the same ontology type and have the same context (possibly empty context). We can also detect schematic discrepancy that is defined below.

Definition 2 *Two schemas are* ***schematically discrepant*** *from each other iff metadata in one schema correspond to attribute values in the other schema. We call the meta-attributes whose values correspond to attribute values in other schemas* ***discrepant meta-attributes****.* □

In Figure 1, $DB1$ and $DB3$ are schematically discrepant from each other, as the metadata 'JAN', ..., 'DEC' of the relationship types JAN_SUP, ..., DEC_SUP in $DB3$ are modelled as the values of the attribute M# in $DB1$. In this case, $m\#$ is a discrepant meta-attribute of the relationship types in $DB3$. Similarly, $DB1$ and $DB2$ are schematically discrepant from each other, and $m\#$ and $s\#$ are discrepant meta-attributes in $DB2$ whose values correspond to attribute values in $DB1$.

4 Resolution of Schematic Discrepancies

In this section, we resolve schematic discrepancies in the integration of ER schemas. In particular, we present four algorithms to resolve schematic discrepancies for entity types, relationship types, attributes of entity types and attributes of relationship types respectively. This is done by transforming discrepant meta-attributes into attributes of entity types. The transformation keeps the cardinalities of attributes and entity types, and therefore preserves FDs and MVDs (Section 5). Note in the presence of context, the values of an attribute depend on not only the key values of the entity type/relationship type, but also the metadata of the attribute.

We present 4 algorithms ResolveEnt, ResolveRel, ResolveEntAttr and ResolveRelAttr, the resolutions of schematic discrepancies for entity types, relationship types, attributes of entity types and attributes of relationship types one by one. Examples are provided to understand each algorithm. Finally, we introduce the general process of integrating ER schemas with different kinds of semantic heterogeneities.

To simplify the presentation, we assume schema constructs only have discrepant meta-attributes, leaving out other meta-attributes that will not cause schematic discrepancies. Actually, non-discrepant meta-attributes will not be changed in schema transformation.

4.1 Resolving Schematic Discrepancies for Entity Types

Given an ER schema, we resolve the schematic discrepancies of the entity types of the schema in 2 steps. In Step 1, we resolve the schematic discrepancies of each entity type, and in Step 2, we merge the equivalent schema constructs in the transformed schema. Step 1 is further divided into 3 sub-steps. Given an entity type E, in Step 1.1, we transform the discrepant meta-attributes of E into the attributes of entity types, and relate the entity types in a relationship type. Then in Step 1.2, we handle the attributes of E according to the ways the attributes inherit the context of E. Finally in Step 1.3, we handle the relationship types involving E according to the ways the relationship types inherit the context of E.

In what follows, we first show two examples of the resolution of schematic discrepancies of entity types, which focus on handling attributes and handling relationship types respectively, and then give the general algorithm.

Example 3 *In DB2 of Figure 1, the entity types JAN_PROD, ..., DEC_PROD have the same discrepant meta-attribute m#. In Figure 3, we resolve the schematic discrepancies of these entity types in two steps.*

In Step 1, for each entity type of DB2, say JAN_PROD = pm[m#='JAN'], we represent the discrepant meta-attribute m# as an attribute M# (with the only value 'JAN') of a new created entity type MONTH = month. As in the ontology, pm is a binary relationship type between the entity types product and month, after removing the context, we change the entity type JAN_PROD into an entity type PROD = product (with all the entities of JAN_PROD), and construct a relationship type PM = pm to associate the entity types PROD and MONTH.

Then we handle the attributes of JAN_PROD. As PNAME has nothing to do with the context of the entity type, it becomes an attribute of PROD. However, S1_PRICE, ..., Sn_PRICE inherit the context m#='JAN', i.e., their values depend on not only the product numbers, but also the month January. So they become the attributes of the relationship type PM. Note as the context of JAN_PROD that is the inherited context of the attributes S1_PRICE, ..., Sn_PRICE is removed, these attributes only have the self context s#='Si' left for i=1, ..., n (the discrepant meta-attribute s# will be resolved in Algorithm ResolveRelAttr later).

Similarly, we can resolve the schematic discrepancies of the other entity types FEB_PROD, ..., DEC_PROD.

Then in Step 2, the equivalent entity types, relationship types and attributes are merged respectively. Their domains are united. □

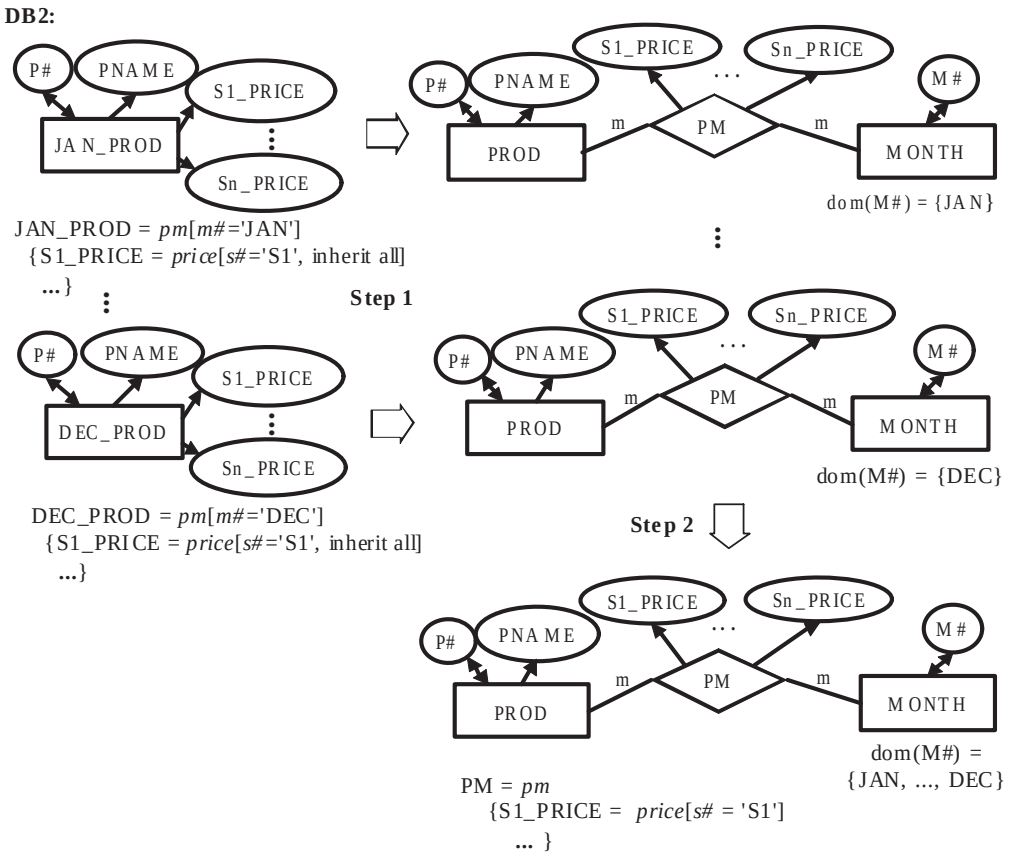


Fig. 3. Resolve schematic discrepancies for entity types: handle attributes

Then we show the other example in which we need to deal with relationship types in the resolution of schematic discrepancies of entity types.

Example 4 *In Figure 4, we give another ER schema DB4 modelling the similar information as those in Figure 1. In DB4, each entity type of JAN_PROD, . . . , DEC_PROD models the products supplied in one month, and each relationship type SUP_i, i=1, . . . , 12, models the supply relationships in the i-th month. Note in DB4, we have a constraint that is none in the schemas of Figure 1: "in each month, a product is uniquely supplied by one supplier." This constraint (i.e., a FD $P\# \rightarrow S\#$) is represented as a cardinality constraint on each relationship type SUP_i.*

In Figure 4, we resolve the schematic discrepancies of the entity types JAN_PROD, . . . , DEC_PROD in 3 steps. In Step 1, for each of these entity type, say JAN_PROD = pm[m#='JAN'], we transform the discrepant meta-attribute m# to an attribute M# of a new created entity type MONTH = month, and connect the entity types PROD and MONTH with a relationship type PM = pm.

Then we handle the relationship type SUP_1 that involves the entity type JAN_PROD.

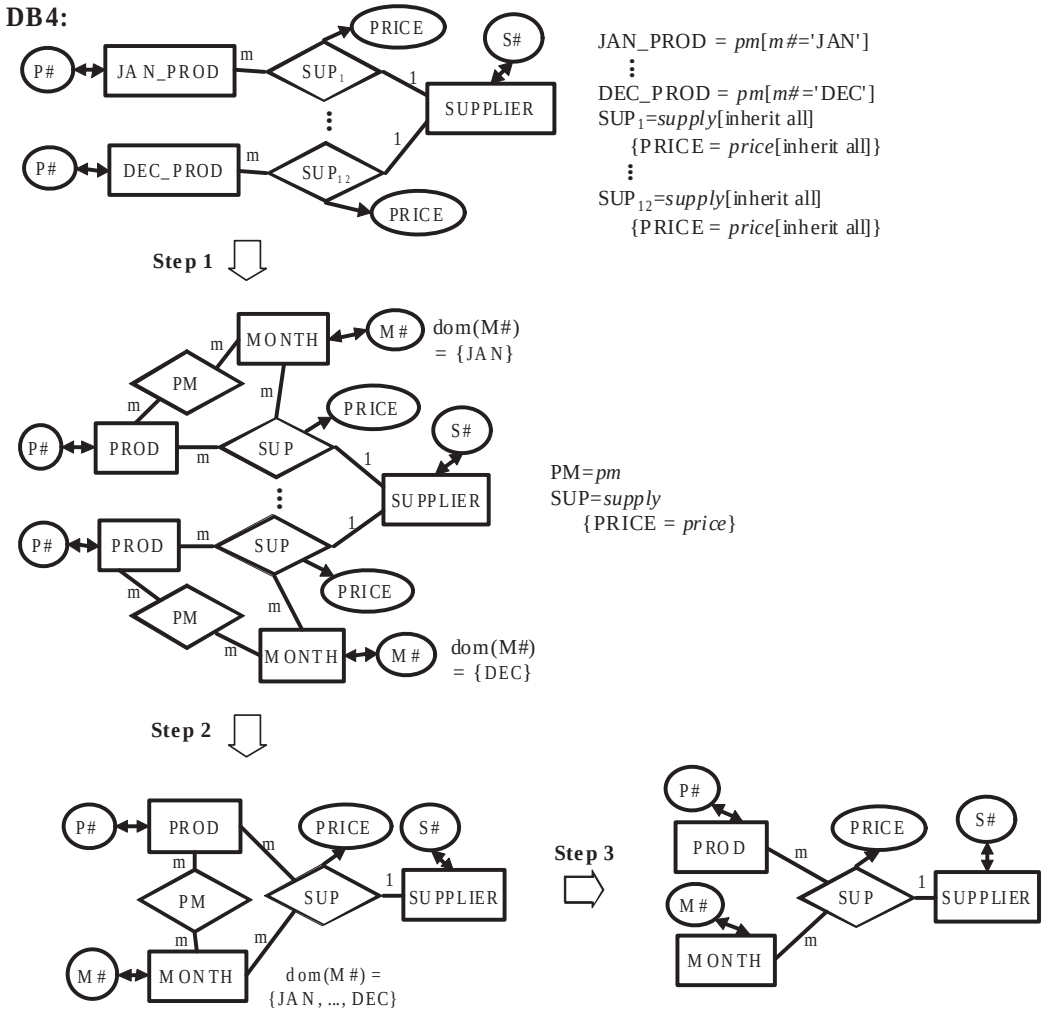


Fig. 4. Resolve schematic discrepancies for entity types: handle relationship types

As SUP_1=supply[inherit all], i.e., it inherits the context m#='JAN' from JAN_PROD, and we have removed the context of JAN_PROD, SUP_1 becomes a ternary relationship type SUP=supply connecting the entity types PROD, MONTH and SUPPLIER.

Similarly, we can transform the entity types FEB_PROD, ..., DEC_PROD and the relationship types SUP_2, ..., SUP_{12}.

Then in Step 2, the equivalent entity types, relationship types and attributes are merged respectively. Their domains are united.

Finally in Step 3, as in the ontology, the relationship type pm is a projection of the relationship type supply, the relationship type PM of the transformed ER schema is redundant and therefore removed. Note that this step is not included in Algorithm ResolveEnt. Instead, it will be performed later in a main integration algorithm calling the resolution algorithms (Section 4.5).

Note that the cardinality constraints on the relationship types SUP_i's of DB4 are represented as an equivalent cardinality constraint (i.e., a FD {P#, M#} → S#) on the relationship type SUP of the transformed schema. This issue will be studied in detail in Section 5. □

The general algorithm is given below.

Algorithm ResolveEnt
Given an ER schema DB, the algorithm produces a schema DB' transformed from DB such that all the discrepant meta-attributes of the entity types are transformed into the attributes of entity types.

Step 1 *Resolve the discrepant meta-attributes of an entity type.*
Let $E = T[C_1 = c_1, \ldots, C_l = c_l, inherit\ C_{l+1}, \ldots, C_m]$ be an entity type of DB, where T is a relationship type among m+1 entity types $T_1, \ldots, T_m, T_{m+1}$ in the ontology, and $C_1, \ldots, C_m$ are m discrepant meta-attributes that are identifiers of $T_1, \ldots, T_m$. Each $C_i, i = 1, \ldots, m$, has a value of c_i. Let C_{m+1} be the identifier of T_{m+1}, such that the identifier of E, $K = C_{m+1}$.

Step 1.1 *Transform $C_1, \ldots, C_m$ into the attributes of entity types.*
Construct m+1 entity types $E_1 = T_1, \ldots, E_m = T_m, E_{m+1} = T_{m+1}$ with the identifiers $K_1 = C_1, \ldots, K_m = C_m, K = C_{m+1}$ (note that the identifier of E_{m+1} is the same as the identifier of E) if they do not exist.
Each E_i (i=1, …, m) contains one entity with the identifier $C_i = c_i$. E_{m+1} contains all the entities of E.
Construct a relationship type $R = T$ connecting $E_1, \ldots, E_{m+1}$, such that $(c_1, \ldots, c_m, k) \in R[K_1, \ldots, K_m, K]$ iff $k \in E[K]$.

Step 1.2 *Handle the attributes of E.*
Let A be an attribute (not part of the identifier) of E. A corresponds to a type A_{ont} in the ontology, and has a self context (i.e., a set of meta-attributes with values) $selfCnt$.
If A is a many-to-one or many-to-many attribute, **then**
Case 1 A does not inherit any context of E.
Then A becomes an attribute of E_{m+1}, such that
$(k, a) \in E_{m+1}[K, A]$ iff $(k, a) \in E[K, A]$.
Case 2 $A = A_{ont}[selfCnt, inherit\ all]$, i.e., A inherits all the context $\{C_1 = c_1, \ldots, C_m = c_m\}$ from E.
Then construct an attribute $A' = A_{ont}[selfCnt]$ of R, such that
$(c_1, \ldots, c_m, k, a) \in R[K_1, \ldots, K_m, K, A']$ iff $(k, a) \in E[K, A]$.
A' has the same cardinality as A.
Case 3 A inherits some context from E. Without losing generality, let $A = A_{ont}[selfCnt, inherit\ C_1, \ldots, C_j]$ for $1 \leq j < m$.
Then construct a relationship type R' connecting E_{m+1} and $E_1, \ldots, E_j$.
Construct an attribute $A' = A_{ont}[selfCnt]$ of R', such that
$(c_1, \ldots, c_j, k, a) \in R'[K_1, \ldots, K_j, K, A']$ iff $(k, a) \in E[K, A]$.
A' has the same cardinality as A.

else /* *A is a one-to-one or one-to-many attribute, i.e., A determines the identifier of E in the context. We keep the inherited context of A, and delay the resolution of it in Algorithm ResolveEntAttr, the resolution for attributes of entity types, in which A will be transformed to the identifier of an entity type to preserve the cardinality constraint.* */

Construct an attribute $A' = A_{ont}[Cnt]$ of E_{m+1}, where Cnt is the self context of A' that is the union of the self and inherited contexts of A, such that

$(k, a) \in E_{m+1}[K, A']$ iff $(k, a) \in E[K, A]$.

Step 1.3 *Handle the relationship types involving the entity type E in DB.*

Let $R1$ be a relationship type involving E in DB, and S be a sequence of the identifiers of all the entity types involved in $R1$. We transform $R1$ into a relationship type $R1'$ as below.

If $R1$ has no attributes, or only has many-to-one and many-to-many attributes, **then**

Case 1 $R1$ does not inherit any context of E.

Then replace E with E_{m+1} in $R1$, and change $R1$ to $R1'$, such that $s \in R1'[S]$ iff $s \in R1[S]$. /**Note that the identifier of E is the same as the identifier of* E_{m+1}.*/

Represent each FD on $R1$ (that is represented as a cardinality constraint of the participating entity types in $R1$) in $R1'$.

Case 2 $R1$ inherits all the context $\{C_1 = c_1, \ldots, C_m = c_m\}$ from E.

Then construct $R1'$ involving $E_1, \ldots, E_m, E_{m+1}$ and all the entity types in $R1$ except E, such that

$(s, c_1, \ldots, c_m) \in R1'[S, K_1, \ldots, K_m]$ iff $s \in R1[S]$.

Let $\mathbb{A} \rightarrow \mathbb{B}$ be a FD on $R1$, where $\mathbb{A}$ and $\mathbb{B}$ are two sets of the identifiers of some participating entity types in $R1$.

If K, the identifier of E, is in $\mathbb{A} \cup \mathbb{B}$, **then**

Represent a FD $\mathbb{A}, K_1, \ldots, K_m \rightarrow \mathbb{B}$ in $R1'$.

Else Represent the same FD $\mathbb{A} \rightarrow \mathbb{B}$ in $R1'$.

Case 3 $R1$ inherits some context, say $\{C_1 = c_1, \ldots, C_j = c_j\}$ $(1 \leq j < m)$ from E.

Then construct $R1'$ involving $E_1, \ldots, E_j$, E_{m+1} and all the entity types in $R1$ except E, such that

$(s, c_1, \ldots, c_j) \in R1'[S, K_1, \ldots, K_j]$ iff $s \in R1[S]$.

Let $\mathbb{A} \rightarrow \mathbb{B}$ be a FD on $R1$, where $\mathbb{A}$ and $\mathbb{B}$ are two sets of the identifiers of some participating entity types in $R1$.

If $K \in \mathbb{A} \cup \mathbb{B}$, **then**

Represent a FD $\mathbb{A}, K_1, \ldots, K_j \rightarrow \mathbb{B}$ in $R1'$.

Else Represent the same FD $\mathbb{A} \rightarrow \mathbb{B}$ in $R1'$.

In each of the 3 cases, $R1'$ and $R1$ have the same attributes, correspond to the same relationship type of the ontology, and have the same self context. $R1'$ has no inherited context.

Else /**R1 has some one-to-one or one-to-many attributes. In order to*

preserve the cardinality constraints of the attributes of $R1$, we keep the inherited context of $R1$ in $R1'$. This context would be removed in Algorithm ResolveRel and ResolveRelAttr later./*
Replace E with E_{m+1} in $R1$, and change $R1$ to $R1'$, such that $s \in R1'[S]$ iff $s \in R1[S]$.
The context of $R1'$ is the union of the self and inherited contexts of $R1$.

Step 2 *Merge equivalent entity types and equivalent relationship types.*
For each set of equivalent entity types $\mathbb{E}$, **do**
Let E be the merged entity type.
The attribute set of E is the union of the attribute sets of all the entity types of $\mathbb{E}$.
For each set of equivalent attributes of some entity types of $\mathbb{E}$, **do**
Resolve the constraint conflicts in the attributes.
*/*Algorithms to resolve constraint conflicts are given in [14].*/*
Unite the domains of these equivalent attributes.
For each set of equivalent relationship types $\mathbb{R}$, **do**
Let R be the merged relationship type.
The attribute set of R is the union of the attribute sets of all the relationship types of $\mathbb{R}$.
Resolve the constraint conflicts in the relationship types.
*/*Algorithms to resolve constraint conflicts are given in [14].*/*
For each set of equivalent attributes of some relationship types of $\mathbb{R}$, **do**
Resolve the constraint conflicts in the attributes.
Unite the domains of these equivalent attributes. □

4.2 Resolving Schematic Discrepancies for Relationship Types

In the resolution of schematic discrepancies for relationship types, we should deal with a set of entity types (participating in a relationship type) instead of individual ones. The resolution can also be performed in 2 steps: first transform the discrepant meta-attributes of relationship types into the attributes of entity types (unlike Algorithm ResolveEnt, we don't need Step 1.3 in Algorithm ResolveRel), and then merge the equivalent schema constructs in the transformed schema. We first present an example below.

Example 5 *In DB3 of Figure 1, the relationship types JAN_SUP, . . . , DEC_SUP have the same discrepant meta-attribute m#. In Figure 5, we resolve the schematic discrepancies of these relationship types in two steps.*

In Step 1, for each relationship type of DB3, say JAN_SUP = supply[m#='JAN'], we represent the meta-attribute m# as an attribute M# of a new created entity type MONTH. After removing the context, we change JAN_SUP into a ternary relationship type SUP = supply, to connect the entity types PROD,

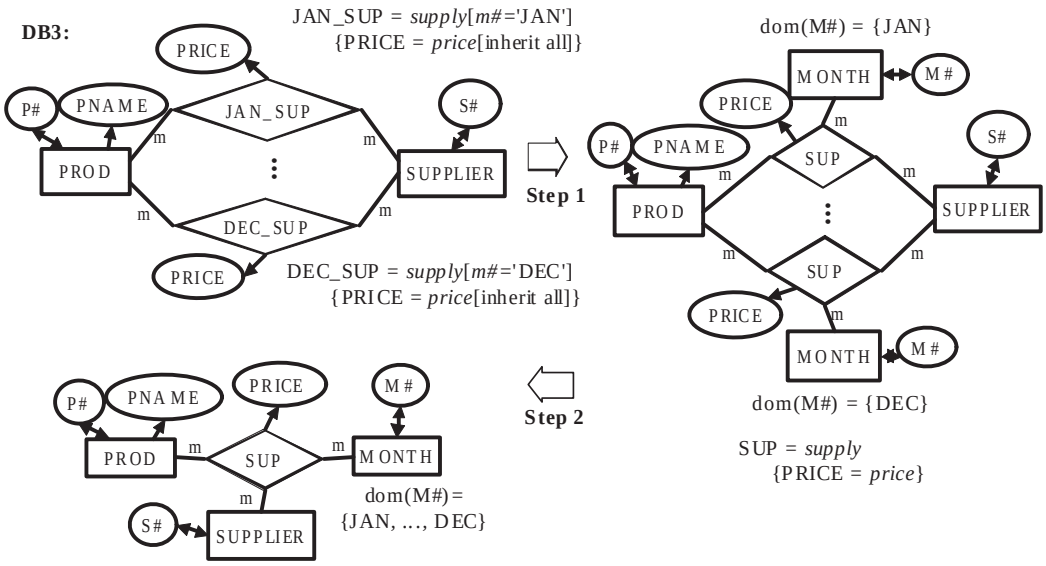


Fig. 5. Resolve schematic discrepancies for relationship types

MONTH and SUPPLIER.

Then we handle the attribute PRICE of the relationship type JAN_SUP. As PRICE = price[inherit all], i.e., its values depend on not only product numbers and supplier numbers, but also months, PRICE = price becomes an attribute of SUP in the transformed schema.

Similarly, we can transform the other relationship types FEB_SUP, . . . , DEC_SUP.

Then in Step 2, the equivalent entity types, relationship types and attributes are merged. Their domains are united. □

The general algorithm ResolveRel is presented in Appendix A.1. Note as the resolution of the schematic discrepancies for relationship types always follows the resolution for entity types, the relationship types input to Algorithm ResolveRel have no inherited context (see Step 1.3 of Algorithm ResolveEnt for the transformation of relationship types in the resolution of the schematic discrepancies of entity types).

4.3 Resolving Schematic Discrepancies for Attributes of Entity Types

Given an ER schema, we resolve the schematic discrepancies of the attributes of entity types in two steps. In Step 1, given an attribute A of an entity type, we transform the discrepant meta-attributes of A into the attributes of entity types, and transform A to an attribute of a relationship type or the identifier of an entity type. Then in Step 2, we merge equivalent schema constructs

of the transformed schema. We first explain the resolution algorithm by an example below.

Example 6 *In Figure 6, we give another ER schema DB5 modelling the similar information as those in Figure 1. In DB5, each of the* $12 \times n$ *attributes S1_JAN_PRICE, . . . , Sn_DEC_PRICE models the prices of the products supplied by one supplier in one month.*

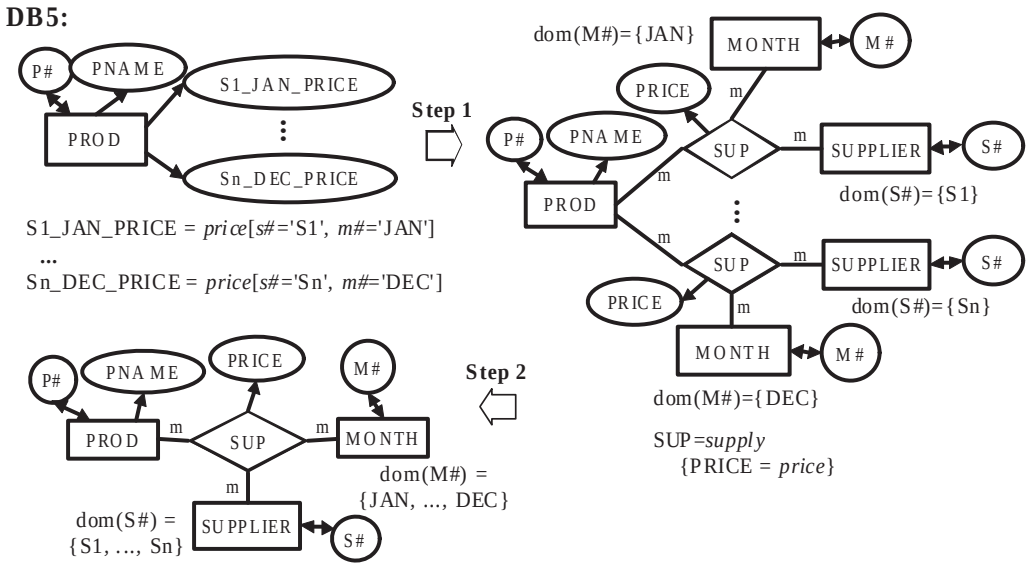


Fig. 6. Resolve schematic discrepancies for attributes of entity types

In Figure 6, we resolve the schematic discrepancies of the attributes S1_JAN_PRICE, . . . , Sn_DEC_PRICE in two steps. In Step 1, for each of the attributes, say S1_JAN_PRICE = price[s#=S1, m#=JAN], we represent the discrepant meta-attributes s# and m# as the attributes of new created entity types SUPPLIER = supplier and MONTH = month. In the ontology, price is an attribute of the ternary relationship type supply. Then in the ER schema, we construct the relationship type SUP = supply to contain the attribute PRICE = price.

Similarly, we can transform the other attributes S1_FEB_PRICE, . . . , Sn_DEC_PRICE.

Then in Step 2, we merge all the equivalent entity types, relationship types and attributes. Their domains are united. □

The general algorithm ResolveEntAttr is presented in Appendix A.2. Note as the resolution of the schematic discrepancies for the attributes of entity types always follows the resolution for entity types, the attributes input to Algorithm ResolveEntAttr have no inherited context (see Step 1.2 of Algorithm ResolveEnt for the transformation of attributes in the resolution of the schematic discrepancies of entity types).

4.4 Resolving Schematic Discrepancies for Attributes of Relationship Types

Given an ER schema, we resolve the schematic discrepancies of the attributes of relationship types in two steps, i.e., Step 1 of transforming the discrepant meta-attributes into the attributes of entity types and Step 2 of merging. Note unlike Algorithm ResolveEntAttr, in Algorithm ResolveRelAttr, we need to deal with a set of entity types involved in a relationship type instead of individual entity types. We first explain the resolution algorithm by an example below.

Example 7 *In the transformed schema of Figure 3, the attributes S1_PRICE, . . . , Sn_PRICE of the relationship type PM represent the prices of the products supplied by the suppliers S1, . . . , Sn in some months. These attributes have the same discrepant meta-attribute s#.*

In Figure 7, we resolve the schematic discrepancies of the attributes S1_PRICE, . . . , Sn_PRICE in three steps.

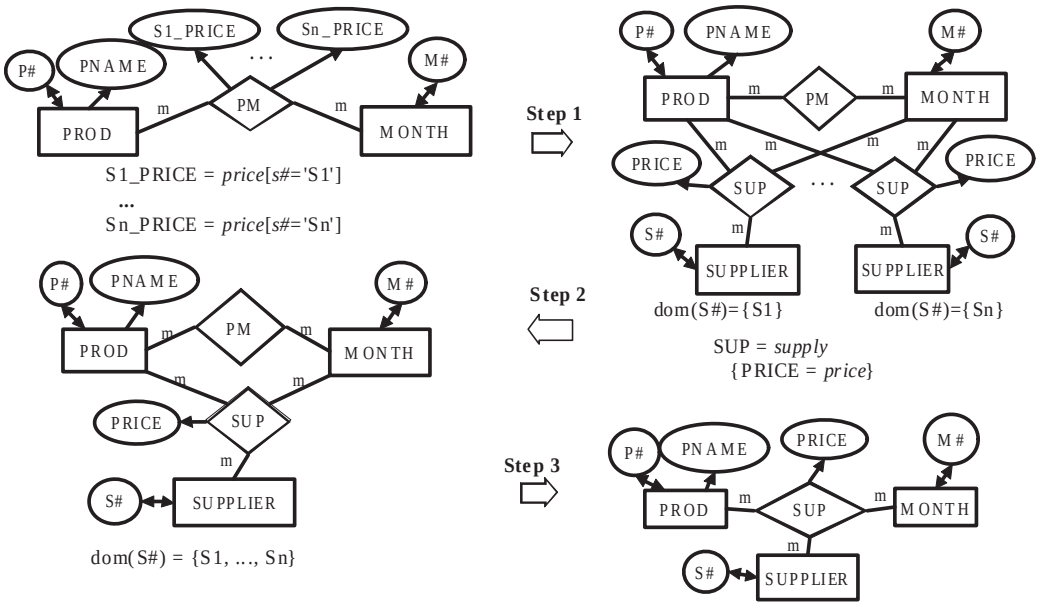


Fig. 7. Resolve schematic discrepancies for attributes of relationship types

In Step 1, for each attribute, say S1_PRICE = price[s#='S1'], we represent the discrepant meta-attribute s# as an attribute S# of a new entity type SUPPLIER = supplier. In the ontology, price is an attribute of the ternary relationship type supply. Then in the ER schema, we construct the relationship type SUP = supply to contain the attribute PRICE = price.

Similarly, we transform the other attributes S2_PRICE, . . . , Sn_PRICE.

Then in Step 2, equivalent entity types, relationship types and attributes are

merged respectively. Their domains are united.

Finally in Step 3, as in the ontology, the binary relationship type pm is a projection of the terary relationship type supply, in the ER schema, the relationship type PM is redundant and therefore removed. Note that this step is not included in Algorithm ResolveRelAttr. Instead, it will be performed later in a main integration algorithm calling the resolution algorithms (Section 4.5). □

The general algorithm ResolveRelAttr is presented in Appendix A.3. Note as the resolution of the schematic discrepancies for the attributes of relationship types always follows the resolution for relationship types, the attributes input to Algorithm ResolveRelAttr have no inherited context (see Step 1.2 of Algorithm ResolveRel for the transformation of attributes in the resolution of schematic discrepancies of relationship types).

4.5 Integration of ER Schemas

In schema integration, the 4 kinds of schematic discrepancies should be resolved in the order of context inheritance presented in Section 3, i.e., first for entity types, then relationship types, finally attributes of entity types and attributes of relationship types. The resolutions of the other semantic heterogeneities follow the resolution of schematic discrepancies. In general, given a set of ER schemas, we can integrate them in 4 steps:

(1) Call the algorithms ResolveEnt, ResolveRel, ResolveEntAttr and ResolveRelAttr in order to resolve the schematic discrepancies of entity types, relationship types, attributes of entity types and attributes of relationship types.
(2) Resolve the other semantic heterogeneities of naming conflicts, key conflicts, structural conflicts, etc, using existing methods, e.g., [15].
(3) Merge the transformed schemas. Equivalent entity types, relationship types and attributes are superimposed. Some constraint conflicts may need to be resolved during the merging [14].
(4) Remove the redundant relationship types which can be derived from the others. Create special relationship types ISA, UNION, INTERSECT or DECOMPOSE among entity types of the integrated schema.

5 Semantics Preserving Transformation

In this section, we will show that Algorithm ResolveEnt (Section 4.1), the resolution of the schematic discrepancies of entity types, preserves information and cardinality constraints. The same property holds for the other three algorithms, which is omitted as the proofs are similar to that of Algorithm ResolveEnt.

5.1 Semantics Preservation of Algorithm ResolveEnt

As to the information preservation, we have the following result, the proof of which is omitted.

Theorem 1 *The transformation of Step 1 of Algorithm ResolveEnt is a one to one mapping from the instance set of the original schema onto the instance set of the transformed schema.* □

This can be concluded from the necessary and sufficient conditions of the data transformation statements in the algorithm (i.e., the "iff" statements in Step 1 of Algorithm ResolveEnt).

Note in the algorithm, Step 2 of merging (implemented using union or outerjoin operations in data integration) is a many to one mapping, which makes the recovery impossible. It seems acceptable, as in an information integration system, sources are usually transparent to users. That is, users usually do not care about the sources from which an object come.

The same result of Theorem 1 holds for the other three algorithms in Appendix, i.e., Algorithm ResolveRel, the resolution of the schematic discrepancies of relationship types, Algorithm ResolveEntAttr, the resolution of the schematic discrepancies of the attributes of entity types, and Algorithm ResolveRelAttr, the resolution of the schematic discrepancies of the attributes of relationship types.

In what follows, we study the preservation of FDs and MVDs in the schema transformation of Algorithm ResolveEnt.

Theorem 2 *The schema transformation of Algorithm ResolveEnt preserves the constraints of FDs and MVDs.* □

We prove this theorem through 5 lemmas in the rest of this section. In ER schemas, cardinality constraints (in particular, the cardinalities of the attributes of entity types, the cardinalities of the entity types in a relationship

type, or the cardinalities of the attributes of relationship types) may represent FDs/MVDs, as mentioned in Section 2.1. Lemma 1 is on the preservation of the FDs and MVDs represented as the cardinality constraints of the attributes of entity types. Lemma 2 and 3 are on the preservation of the FDs represented as the cardinality constraints of the entity types in a relationship type. Lemma 4 and 5 are on the preservation of the FDs and MVDs represented as the cardinality constraints of the attributes of a relationship type.

We first show an example of the preservation of the FDs that are the cardinality constraints of the attributes of entity types below.

Example 8 *In Figure 3, in each of the entity types* $JAN\_PROD, \ldots, DEC\_PROD$ *of DB2, the attributes* $S1\_PRICE, \ldots, Sn\_PRICE$ *inherit all the context of the entity type. After Algorithm ResolveEnt, the discrepant meta-attribute* $m\#$ *becomes the attribute* $M\#$ *of the entity type MONTH in the transformed schema, and* $S1\_PRICE, \ldots, Sn\_PRICE$ *become the attributes of the relationship type* PM*. We have the following result:*

A FD $P\# \rightarrow \{S1\_PRICE, \ldots, Sn\_PRICE\}$ *holds in each entity type of DB2 iff a FD* $\{P\#, M\#\} \rightarrow \{S1\_PRICE, \ldots, Sn\_PRICE\}$ *holds in the relationship type* PM *of the transformed schema.*

On the other hand, in each entity type of DB2, the attribute $PNAME$ *has nothing to do with the context of the entity type, i.e., a product name is only dependent on the product number, independent of the months in which the product is supplied. We have the following result:*

A FD $P\# \rightarrow PNAME$ *holds in each entity type of DB2 iff the same FD* $P\# \rightarrow PNAME$ *holds in the entity type* $PROD$ *of the transformed schema.* □

In general, we have the following result:

Lemma 1 *Algorithm ResolveEnt preserves the FDs and MVDs represented as the cardinality constraints of the attributes of entity types.*

Proof: Recall Step 1.2 of Algorithm ResolveEnt in which we transform the attributes of entity types in the resolution of schematic discrepancies of entity types. In what follows, we first claim that for each case of Step 1.2, the cardinality constraints of attributes can be preserved in the transformed schema, then prove a typical case. In Algorithm ResolveEnt, equivalent schema constructs will be merged in Step 2. Without losing generality, we consider a set of entity types (instead of individual ones) that correspond to the same ontology type, and have the same set of meta-attributes but different metadata, such as the entity types $JAN\_PROD, \ldots, DEC\_PROD$ of $DB2$ in the above example. Such entity types will be transformed to equivalent relationship types

and merged in the algorithm.

In general, in an ER schema DB, let $\mathbb{E}$ be a set of entity types with the same identifier K, the same attribute A and the same meta-attributes, i.e.,

$$\mathbb{E} = \{E | E = T[C_1 = c_1, \ldots, C_l = c_l, inherit\ C_{l+1}, \ldots, C_m], \\ c_1 \in dom(C_1), \ldots, c_l \in dom(C_l)\}.$$

Let A correspond to an attribute A_{ont} in the ontology, and have a self context $selfCnt$, i.e., a set of meta-attributes with values.

Let DB' be the schema transformed from DB by Algorithm ResolveEnt, in which $C_1, \ldots, C_m$ become the identifiers $K_1, \ldots, K_m$ of entity types $E_1, \ldots, E_m$, and an entity type E_{m+1} with the identifier K is created to contain all the entities of the entity types of $\mathbb{E}$. We claim that:

Case 1 A is a many-to-one or many-to-many attribute in each entity type of $\mathbb{E}$.

Case 1.1 $A = A_{ont}[selfCnt]$ does not inherit any context from the entity types.

Then A becomes a non-identifier attribute of the entity type E_{m+1} in the transformed schema DB'.

If A is a many-to-one attribute, **then**

A FD $K \rightarrow A$ holds in each entity type of $\mathbb{E}$ iff a FD $K \rightarrow A$ holds in E_{m+1}.

Else /*A *is a many-to-many attribute.**/

A MVD $K \twoheadrightarrow A$ holds in each entity type of $\mathbb{E}$ iff a MVD $K \twoheadrightarrow A$ holds in E_{m+1}.

Case 1.2 $A = A_{ont}[selfCnt, inherit\ all]$ inherits all the context of the entity types.

Then $A' = A_{ont}[selfCnt]$ becomes an attribute of a new-created relationship type R among m+1 entity types $E_1, \ldots, E_{m+1}$ in DB'.

If A is a many-to-one attribute, **then**

A FD $K \rightarrow A$ holds in each entity type of $\mathbb{E}$ iff a FD $K_1, \ldots, K_m, K \rightarrow A'$ holds in R.

Else /*A *is a many-to-many attribute.**/

A MVD $K \twoheadrightarrow A$ holds in each entity type of $\mathbb{E}$ iff a MVD $K_1, \ldots, K_m, K \twoheadrightarrow A'$ holds in R.

Case 1.3 $A = A_{ont}[selfCnt, inherit\ C_1, \ldots, C_j]$ $(1 \leq j < m)$ inherits some context of the entity types.

Then $A' = A_{ont}[selfCnt]$ becomes an attribute of a new-created relationship type R' among j+1 entity types $E_1, \ldots, E_j$ and E_{m+1} in DB'.

If A is a many-to-one attribute, **then**

A FD $K \rightarrow A$ holds in each entity type of $\mathbb{E}$ iff a FD $K_1, \ldots, K_j, K \rightarrow A'$ holds in R'.

Else /*A *is a many-to-many attribute.*/
A MVD $K \twoheadrightarrow A$ holds in each entity type of $\mathbb{E}$ iff a MVD $K_1, \ldots, K_j, K \twoheadrightarrow A'$ holds in R'.

Case 2 A is a one-to-one or one-to-many attribute.
Then $A' = A_{ont}[Cnt]$ becomes an attribute of E_{m+1} in DB', where Cnt is the self context of A' that is the union of the self context (i.e., $selfCnt$) and inherited context (say $inhrtCnt$) of A.
A FD $A \rightarrow K$ holds in each entity type of $\mathbb{E}$ which has the context $inhrtCnt$ iff $A' \rightarrow K$ holds in E_{m+1}.
If A is a one-to-one attribute, **then**
A FD $K \rightarrow A$ holds in each entity type of $\mathbb{E}$ which has the context $inhrtCnt$ iff a FD $K \rightarrow A'$ holds in E_{m+1}.
Else A MVD $K \twoheadrightarrow A$ holds in each entity type of $\mathbb{E}$ which has the context $inhrtCnt$ iff a MVD $K \twoheadrightarrow A'$ holds in E_{m+1}.

Then we prove the case when A is a many-to-one attribute that inherits all the context of the entity types, i.e, a sub-case of Case 1.2. The other cases can be proven in a similar way.

(=>) If a FD $K \rightarrow A$ holds in each entity type of $\mathbb{E}$, then a FD $K_1, \ldots, K_m, K \rightarrow A'$ holds in R.
Suppose we are given two tuples $(c_1, \ldots, c_m, k, a), (c_1, \ldots, c_m, k, a') \in R[K_1, \ldots, K_m, K, A']$. As the two tuples have the same values on $K_1, \ldots, K_m$ (i.e., $C_1, \ldots, C_m$), they must come from the same entity type of $\mathbb{E}$. As a FD $K \rightarrow A$ holds in each entity type of $\mathbb{E}$, we have $a = a'$. Consequently, $K_1, \ldots, K_m, K \rightarrow A'$ holds in R.

(<=) If a FD $K_1, \ldots, K_m, K \rightarrow A'$ holds in R, then a FD $K \rightarrow A$ holds in each entity type of $\mathbb{E}$.
For each entity type $E = T[C_1 = c_1, \ldots, C_l = c_l, inherit\ C_{l+1}, \ldots, C_m]$ in $\mathbb{E}$, given two tuples $(k, a), (k, a') \in E[K, A]$, by Algorithm ResolveEnt, we can transform them to two tuples $(c_1, \ldots, c_m, k, a), (c_1, \ldots, c_m, k, a') \in R[K_1, \ldots, K_m, K, A']$. As a FD $K_1, \ldots, K_m, K \rightarrow A'$ holds in R, we have $a = a'$. Consequently, $K \rightarrow A$ holds in E. □

In the ER approach, FDs/MVDs can be represented by not only the cardinalities of the attributes of entity types, but also the cardinalities of the entity types and the cardinalities of the attributes in a relationship type. In what follows, we first show an example of FDs represented as the cardinality constraints in relationship types, then present 4 lemmas to generalize the results.

Example 9 *In the schema DB4 of Figure 4, a relationship type* SUP_1 *(the other relationship types are similar) inherits the context m#='JAN' from its participating entity type JAN_PROD. After Algorithm ResolveEnt, the discrepant meta-attribute m# becomes an attribute M# of the entity type MONTH*

in the transformed schema, and the relationship types SUP_i's are transformed and merged into a ternary relationship type SUP among the entity types PROD, MONTH and SUPPLIER. We have the following result:

A FD $P\# \rightarrow S\#$ holds in each relationship type of DB4 iff a FD $\{P\#, M\#\} \rightarrow S\#$ holds in the relationship type SUP of the transformed schema.

In the schema of DB4, the relationship type SUP_1 (the other relationship types are similar) has an attribute PRICE with the inherited context $m\# = `JAN'$. After Algorithm ResolveEnt, PRICE becomes an attribute of the relationship type SUP in the transformed schema. We have the following result:

A FD $P\# \rightarrow PRICE$ holds in each relationship type of DB4 iff a FD $\{P\#, M\#\} \rightarrow PRICE$ holds in the relationship type SUP of the transformed schema. □

Recall Step 1.3 of Algorithm ResolveEnt in which we transform relationship types in the resolution of schematic discrepancies for entity types. According to whether a relationship type has any one-to-one or one-to-many attributes, the transformation methods would be different. Correspondingly, when presenting the issue of FD preservation, we also divide the two cases. In particular, Lemma 2 and Lemma 4 are, respectively, on the preservation of the cardinalities of entity types and the cardinalities of attributes when relationship types only have many-to-one and many-to-many attributes. On the other hand, Lemma 3 and Lemma 5 are, respectively, on the preservation of the cardinalities of entity types and the cardinalities of attributes when relationship types have some one-to-one or one-to-many attributes.

Lemma 2 *When relationship types only have many-to-one and many-to-many attributes, Algorithm ResolveEnt preserves the FDs represented as the cardinality constraints of the entity types in the relationship types.*

Proof: In what follows, we first claim that for any of the three cases of Step 1.3 in Algorithm ResolveEnt, the cardinality constraints of the entity types in a relationship type can be preserved in the transformed schema. Note a relationship type may involve several entity types with discrepant meta-attributes all of which need to be resolved. For ease of presentation, we will not distinguish the three cases as in Step 1.3 of the algorithm, but rather generalize the cases. In particular, we give the general form of a FD on a relationship type of a transformed schema and the corresponding FDs in the original schema, and show that the FD of the transformed schema and the FDs of the original schema are equivalent to each other. In Step 2 of Algorithm ResolveEnt, equivalent schema constructs will be merged. Without losing generality, we consider a set of relationship types that correspond to the same ontology type, and have the same self context but not necessary the same inherited context. Such relationship types will be transformed to equivalent relationship types and merged in

Algorithm ResolveEnt.

In general, in an ER schema DB, let $\mathbb{R} = \{R_1, \ldots, R_n\}$ be a set of relationship types corresponding to the same ontology type within the same self context, such that each relationship type has no attributes, or only has many-to-one and many-to-many attributes.
Let DB' be the schema transformed from DB by Algorithm ResolveEnt, in which all the relationship types of $\mathbb{R}$ are transformed and merged into a relationship type R'. We claim that:

A FD $K_1, \ldots, K_m \rightarrow K_{m+1}$ holds in R' for $K_1, \ldots, K_{m+1}$ the identifiers of the $m+1$ entity types involved in R', iff in each relationship type $R_i \in \mathbb{R}$, a FD $K_1^i, \ldots, K_l^i$
$\rightarrow K_{l+1}^i$ holds for $K_1^i, \ldots, K_{l+1}^i$ the identifiers of the entity types $E_1^i, \ldots, E_{l+1}^i$ involved in R_i, such that:

(1) K_{m+1} is equivalent to K_{l+1}^i;
(2) for each $j = 1, \ldots, m$, $K_j \in \{K_1^i, \ldots, K_l^i\}$, i.e., K_j is equivalent to some element of $\{K_1^i, \ldots, K_l^i\}$, or K_j corresponds to a discrepant meta-attribute of an entity type of $E_1^i, \ldots, E_{l+1}^i$;
(3) for each $j = 1, \ldots, l+1$, $K_j^i \in \{K_1, \ldots, K_{m+1}\}$, and all the discrepant meta-attributes of E_j^i are represented as the identifiers in $\{K_1, \ldots, K_m\}$.

Then we prove the above claim:
(=>) If a FD $K_1, \ldots, K_m \rightarrow K_{m+1}$ holds in R', then a FD $K_1^i, \ldots, K_l^i \rightarrow K_{l+1}^i$ holds in each relationship type $R_i \in \mathbb{R}$.
Suppose we are given two tuples $(k_1^i, \ldots, k_l^i, k_{l+1}^i), (k_1^i, \ldots, k_l^i, k_{l+1}^i{}') \in R_i[K_1^i, \ldots, K_l^i, K_{l+1}^i]$. These two tuples correspond to $(k_1, \ldots, k_m, k_{m+1}), (k_1, \ldots, k_m, k'_{m+1}) \in R'[K_1, \ldots, K_m, K_{m+1}]$, which satisfy the three conditions of the above claim. As the FD $K_1, \ldots, K_m \rightarrow K_{m+1}$ holds in R', $k_{m+1} = k'_{m+1}$. As k_{m+1} is equivalent to k_{l+1}^i and k'_{m+1} is equivalent to $k_{l+1}^i{}'$, $k_{l+1}^i = k_{l+1}^i{}'$. So a FD $K_1^i, \ldots, K_l^i \rightarrow K_{l+1}^i$ holds in each relationship type R_i.

(<=) If a FD $K_1^i, \ldots, K_l^i \rightarrow K_{l+1}^i$ holds in each relationship type $R_i \in \mathbb{R}$, then a FD $K_1, \ldots, K_m \rightarrow K_{m+1}$ holds in R'.
Suppose we are given two tuples $(k_1, \ldots, k_m, k_{m+1}), (k_1, \ldots, k_m, k'_{m+1}) \in R'[K_1, \ldots, K_m, K_{m+1}]$. These two tuples correspond to $(k_1^i, \ldots, k_l^i, k_{l+1}^i), (k_1^i, \ldots, k_l^i, k_{l+1}^i{}') \in R_i[K_1^i, \ldots, K_l^i, K_{l+1}^i]$ for some relationship type $R_i \in \mathbb{R}$, which satisfy the three conditions of the above claim. As the FD $K_1^i, \ldots, K_l^i \rightarrow K_{l+1}^i$ holds in the relationship type R_i, $k_{l+1}^i = k_{l+1}^i{}'$. As k_{m+1} is equivalent to k_{l+1}^i and k'_{m+1} is equivalent to $k_{l+1}^i{}'$, $k_{m+1} = k'_{m+1}$. So the FD $K_1, \ldots, K_m \rightarrow K_{m+1}$ holds in R'. $\square$

Note in the proof of Lemma 2, two relationship types of $\mathbb{R}$ may involve different sets of entity types because of the interplay of data and metadata (but after Algorithm ResolveEnt, these relationship types will be transformed to

equivalent ones). Consequently, a FD on R' may correspond to different FDs on the relationship types of $\mathbb{R}$.

Lemma 3 *When relationship types have some one-to-one or one-to-many attributes, Algorithm ResolveEnt preserves the FDs represented as the cardinality constraints of the entity types in the relationship types.*

Proof: When a relationship type has some one-to-one or one-to-many attributes, the inherited context of the relationship type would be kept in the transformed relationship type (see Step 1.3 of Algorithm ResolveEnt). In this case, a set of relationship types corresponding to the same ontology type within the same self context would not be necessarily transformed into equivalent relationship types. Further, they should also have the same inherited context to be merged.

In general, in an ER schema DB, let $\mathbb{R} = \{R_1, \ldots, R_n\}$ be a set of relationship types corresponding to the same ontology type within the same context, such that each relationship type has some one-to-one or one-to-many attributes. Let DB' be the schema transformed from DB by Algorithm ResolveEnt, in which all the relationship types of $\mathbb{R}$ are transformed and merged into a relationship type R'. We claim that:

A FD $\mathbb{A} \rightarrow \mathbb{B}$ holds in each relationship type of $\mathbb{R}$ for $\mathbb{A}$ and $\mathbb{B}$ two distinct sets of the identifiers of some entity types involved in each relationship type of $\mathbb{R}$ iff the same FD $\mathbb{A} \rightarrow \mathbb{B}$ holds in R'.

The proof of the claim is omitted. □

Lemma 4 *When relationship types only have many-to-one and many-to-many attributes, Algorithm ResolveEnt preserves the FDs and MVDs represented as the cardinality constraints of the attributes of the relationship types.*

Proof: If a relationship type only has many-to-one and many-to-many attributes, given an attribute of the relationship type, Algorithm ResolveEnt will remove some of its context inherited from the entity types involved in the relationship type, and move the attribute to a new relationship type (see Step 1.3 of Algorithm ResolveEnt). As long as we keep the cardinality of the attribute, the FD/MVD are also preserved, but may be represented in different forms. Note in a relationship type, although the cardinalities of entity types can only represent FDs, the cardinalities of attributes can represent FDs (if they are many-to-one attributes) and MVDs (if they are many-to-many attributes).

In general, in an ER schema DB, let $\mathbb{R} = \{R_1, \ldots, R_n\}$ be a set of relationship types that correspond to the same ontology type, have the same self context and the same attribute A, such that each relationship type only has many-to-

one and many-to-many attributes.
Let DB' be the schema transformed from DB by Algorithm ResolveEnt, in which all the relationship types of $\mathbb{R}$ are transformed and merged into a relationship type R' with the attribute A. We claim:

A FD $K_1, \ldots, K_m \rightarrow A$ (or a MVD $K_1, \ldots, K_m \twoheadrightarrow A$) holds in R' for $K_1, \ldots, K_m$ the identifiers of the m entity types involved in R', iff in each relationship type $R_i \in \mathbb{R}$, a FD $K_1^i, \ldots, K_l^i \rightarrow A$ (or a MVD $K_1^i, \ldots, K_l^i \twoheadrightarrow A$) holds for $K_1^i, \ldots, K_l^i$ the identifiers of the entity types $E_1^i, \ldots, E_l^i$ involved in R_i, such that:

(1) for each $j = 1, \ldots, m$, $K_j \in \{K_1^i, \ldots, K_l^i\}$, or K_j corresponds to a discrepant meta-attribute of an entity type of $E_1^i, \ldots, E_l^i$;
(2) for each $j = 1, \ldots, l$, $K_j^i \in \{K_1, \ldots, K_m\}$, and all the discrepant meta-attributes of E_j^i are represented as the identifiers in $\{K_1, \ldots, K_m\}$.

The proof of the claim is omitted. □

Lemma 5 *When relationship types have some one-to-one or one-to-many attributes, Algorithm ResolveEnt preserves the FDs and MVDs represented as the cardinality constraints of the attributes of the relationship types.*

Proof: If a relationship type R has some one-to-one or one-to-many attributes, we should consider two kinds of dependencies: the identifier of R determines an attribute of R (i.e., a FD or MVD), and the attribute determines the identifier of R (i.e., a FD).

In general, in an ER schema DB, let $\mathbb{R} = \{R_1, \ldots, R_n\}$ be a set of relationship types that correspond to the same ontology type, have the same context and the same attribute A, such that each relationship type has some one-to-one or one-to-many attributes.
Let DB' be the schema transformed from DB by Algorithm ResolveEnt, in which all the relationship types of $\mathbb{R}$ are transformed and merged into a relationship type R' with the attribute A. We claim:

A FD $K \rightarrow A$ (or a MVD $K \twoheadrightarrow A$) for K the identifier of each relationship type of $\mathbb{R}$ holds in each relationship type of $\mathbb{R}$ iff the same FD $K \rightarrow A$ (or the same MVD $K \twoheadrightarrow A$) holds in R'.

Furthermore, If A is a one-to-one or one-to-many attribute, we also have a result below:

A FD $A \rightarrow K$ for K the identifier of each relationship type of $\mathbb{R}$ holds in each relationship type of $\mathbb{R}$ iff the same FD $A \rightarrow K$ holds in R'.

The proof of the claim is omitted. □

This completes the proof of Theorem 2. In a similar way, we can prove that any of the other three algorithms in Appendix, i.e., ResolveRel, ResolveEntAttr and ResolveRelAttr, preserves FDs and MVDs.

6 Related Work

Context is the key component in capturing the semantics related to the definition of an entity type, a relationship type or an attribute. The definition of context as a set of meta-attributes with values is originally adopted in [6] [20], but is used to solve different kinds of semantic heterogeneities. Our work complements rather than competes with theirs. Further, their work is based on the context at the attribute level only. We consider the contexts at different levels, and the inheritance of contexts.

A special kind of schematic discrepancy has been studied in multidatabase interoperability, e.g. [10] [11] [13]. They dealt with the discrepancy when schema labels (e.g., relation names or attribute names) in one database correspond to attribute values in another. However, we use context to capture meta-information, and solve a more general problem in the sense that schema constructs could have multiple (instead of single) discrepant meta-attributes. Furthermore, their work is at the "structure level", i.e., they did not consider the constraint issue in the resolution of schematic discrepancies. However, the importance of constraints can never be overestimated in both individual and multidatabase systems. In particular, we preserve FDs and MVDs during schema transformation, which are expressed as cardinality constraints in ER schemas. The purposes are also different. Previous work focused on the development of multidatabase languages by which users can query over relation names, attribute names and values in relational databases. However, we try to develop an integration system which can detect and resolve schematic discrepancies automatically given the meta-information on source schemas.

The issue of inferring view dependencies (FDs, MVDs, etc) was introduced in [1] [7]. However, their work is based on the views defined using the relational algebra. In other words, they did not solve the inference problem for schematically discrepant views. In [14] [19] [23], people have begun to focus on the derivation of the constraints for integrated schemas from the constraints of component schemas in schema integration. However, their work did not consider schematic discrepancy in schema integration. Our work complements theirs.

7 Conclusion and Future Work

Information integration provides a competitive advantage to businesses, and becomes a major area of investment by software companies today [17]. In this paper, we resolve a common problem in schema integration, schematic discrepancy in general, using the paradigm of context. We define context as a set of meta-attributes with values, which could be at the levels of databases, entity types, relationship types, and attributes. We design algorithms to resolve schematic discrepancies by transforming discrepant meta-attributes into attributes of entity types. The transformation preserves information and FDs/MVDs which are useful in verifying lossless schema transformation, schema normalization and query optimization in multidatabase systems.

We have implemented a schema integration tool to semi-automatically integrate the discrepant schemas of relational databases. Next, we'll try to extend our system to integrate the databases in different models and semi-structured data.

A Resolution Algorithms of Schematic Discrepancies

A.1 Resolving Schematic Discrepancies for Relationship Types

Algorithm ResolveRel
Given an ER schema DB, the algorithm produces a schema DB' transformed from DB such that all the discrepant meta-attributes of relationship types are transformed into the attributes of entity types.

Step 1 *Resolve the discrepant meta-attributes of a relationship type.*
Let $R = T[C_1 = c_1, \ldots, C_m = c_m]$ be a relationship type among entity types $E_{m+1}, E_{m+2}, \ldots, E_n$ in DB, where T is a relationship type among n entity types $T_1, \ldots, T_n$ in the ontology, $C_1, \ldots, C_m$ are m discrepant meta-attributes that are identifiers of $T_1, \ldots, T_m$, and each $E_i = T_i$ for $i = m+1, \ldots, n$ has an identifier $K_i = C_i$. Let T' (a projection of T) be a relationship type among the entity types $T_{m+1}, \ldots, T_n$ in the ontology. /* *Note that the inherited context of R has been removed in Algorithm ResolveEnt if any.*/

Step 1.1 *Transform $C_1, \ldots, C_m$ into attributes of entity types.*
Construct m entity types $E_1 = T_1, \ldots, E_m = T_m$ with the identifiers $K_1 = C_1, \ldots, K_m = C_m$ if they do not exist.
Each E_i $(i = 1, \ldots, m)$ contains one entity with the identifier $C_i = c_i$.
Construct a relationship type $R' = T$ connecting $E_1, \ldots, E_n$, such that

$(c_1, \ldots, c_m, c_{m+1}, \ldots, c_n) \in R'[K_1, \ldots, K_m, K_{m+1}, \ldots, K_n]$ iff
$(c_{m+1}, \ldots, c_n) \in R[K_{m+1}, \ldots, K_n]$.

Let $\mathbb{A} \rightarrow \mathbb{B}$ be a FD on R, where $\mathbb{A}$ and $\mathbb{B}$ are two sets of the identifiers of some participating entity types in R. Represent a FD $\mathbb{A}, K_1, \ldots, K_m \rightarrow \mathbb{B}$ in R'.

Step 1.2 *Handle the attributes of R.*

Let A be an attribute of R. A corresponds to A_{ont} in the ontology, and has a self context $selfCnt$.

If A is a many-to-one or many-to-many attribute, **then**

Case 1 attribute A does not inherit any context of R.

Then A becomes an attribute of a new relationship type $R'' = T'$ among $E_{m+1}, \ldots, E_n$, such that
$(c_{m+1}, \ldots, c_n, a) \in R''[K_{m+1}, \ldots, K_n, A]$ iff
$(c_{m+1}, \ldots, c_n, a) \in R[K_{m+1}, \ldots, K_n, A]$.

Case 2 attribute $A = A_{ont}[selfCnt, inherit\ all]$ inherits all the context $\{C_1 = c_1, \ldots, C_m = c_m\}$ from R.

Then construct an attribute $A' = A_{ont}[selfCnt]$ of R', such that
$(c_1, \ldots, c_m, c_{m+1}, \ldots, c_n, a) \in R'[K_1, \ldots, K_m, K_{m+1}, \ldots, K_n, A']$ iff
$(c_{m+1}, \ldots, c_n, a) \in R[K_{m+1}, \ldots, K_n, A]$.
A' has the same cardinality as A.

Case 3 A inherits some context, say $\{C_1 = c_1, \ldots, C_j = c_j\}$ $(1 \leq j < m)$ from R.

Then construct a relationship type R'' connecting the entity types $E_1, \ldots, E_j, E_{m+1}, \ldots, E_n$.
Construct an attribute $A' = A_{ont}[selfCnt]$ of R'', such that
$(c_1, \ldots, c_j, c_{m+1}, \ldots, c_n, a) \in R''[K_1, \ldots, K_j, K_{m+1}, \ldots, K_n, A']$ iff
$(c_{m+1}, \ldots, c_n, a) \in R[K_{m+1}, \ldots, K_n, A]$.
A' has the same cardinality as A.

else /* *A is a one-to-one or one-to-many attribute, i.e., A determines the identifier of R in the context. We keep the inherited context of A, and delay the resolution of it in Algorithm ResolveRelAttr, in which A will be transformed to the identifier of an entity type to preserve the cardinality constraint.* */
Construct an attribute $A' = A_{ont}[Cnt]$ of the relationship type $R'' = T'$, where Cnt is the self context of A' that is the union of the self and inherited contexts of A, such that
$(c_{m+1}, \ldots, c_n, a) \in R''[K_{m+1}, \ldots, K_n, A']$ iff
$(c_{m+1}, \ldots, c_n, a) \in R[K_{m+1}, \ldots, K_n, A]$.

Step 2 Merge equivalent entity types, relationship types and attributes respectively. Their domains are united. □

Algorithm ResolveEntAttr
Given an ER schema DB, the algorithm produces a schema DB' transformed from DB such that all the discrepant meta-attributes of the attributes of the entity types in DB are transformed into the attributes of entity types.

Step 1 *Resolve the discrepant meta-attributes of an attribute of an entity type.*
Given an entity type $E = T$ (with the identifier K) of DB, let $A = A_{ont}[C_1 = c_1, \ldots, C_m = c_m]$ be an attribute of E, for A_{ont} an attribute of a relationship type T_R, and $C_1, \ldots, C_m$ the discrepant meta-attributes that are identifers of entity types $T_1, \ldots, T_m$ in the ontology. T_R is a relationship type among $T_1, \ldots, T_m$ and T in the ontology. /* *Note that the inherited context of A has been removed in Algorithm ResolveEnt if any.**/
Construct an entity type $E_i = T_i$ with the identifier $K_i = C_i$ for each $i = 1, \ldots, m$ if they do not exist. Each E_i contains one entity with the identifier $C_i = c_i$.
If A is a many-to-one or many-to-many attribute, **then**
Construct a relationship type $R = T_R$ connecting the entity types $E_1, \ldots, E_m$ and E.
Attribute $A' = A_{ont}$ becomes an attribute of R, such that
$(c_1, \ldots, c_m, k, a) \in R[K_1, \ldots, K_m, K, A']$ iff $(k, a) \in E[K, A]$.
else /*A *is a one-to-one or one-to-many attribute, i.e., A and the meta-attributes $C_1, \ldots, C_m$ together determine the identifier K of E. A should be modelled as the identifier of an entity type to preserve the cardinality constraint.**/
Construct $E_{A'}$ with the identifier $A' = A_{ont}$.
Construct a relationship type R' connecting the entity types $E_1, \ldots, E_m$, E and $E_{A'}$, such that
$(c_1, \ldots, c_m, k, a) \in R[K_1, \ldots, K_m, K, A']$ iff $(k, a) \in E[K, A]$.
Represent a FD $K_1, \ldots, K_m, A' \rightarrow K$ as the cardinality constraint on R.
If A is a one-to-one attribute, also represent a FD $K_1, \ldots, K_m, K \rightarrow A'$ on R.
Step 2 Merge equivalent entity types, relationship types and attributes respectively, and unite their domains. □

A.3 Resolving Schematic Discrepancies for Attributes of Relationship Types

Algorithm ResolveRelAttr
Given an ER schema DB, the algorithm produces a schema DB' transformed from DB such that all the discrepant meta-attributes of the attributes of the

relationship types in DB are transformed into the attributes of entity types.

Step 1 *Resolve the discrepant meta-attributes of an attribute of a relationship type.*

In DB, let R (with the identifier K_R) be a relationship type among m entity types $E_1 = T_1, \ldots, E_m = T_m$ with the identifiers $K_1 = C_1, \ldots, K_m = C_m$, and let $A = A_{ont}[C_{m+1} = c_{m+1}, \ldots, C_n = c_n]$ be an attribute of R, where $C_{m+1}, \ldots, C_n$ are discrepant meta-attributes that are identifers of entity types $T_{m+1}, \ldots, T_n$, and A_{ont} is an attribute of a relationship type T among n entity types $T_1, \ldots, T_n$ in the ontology. /* *Note that the inherited context of A has been removed in Algorithm ResolveRel if any.*/

Construct an entity types $E_i = T_i$ with an identifier $K_i = C_i$ for each $i = m+1, \ldots, n$ if it does not exist. Each E_i contains one entity with the identifier c_i.

If A is a many-to-one or many-to-many attribute, **then**

Construct a relationship type $R' = T$ connecting the entity types $E_1, \ldots, E_n$.

Attribute $A' = A_{ont}$ becomes an attribute of R', such that

$(c_1, \ldots, c_m, c_{m+1}, \ldots, c_n, a) \in R'[K_1, \ldots, K_m, K_{m+1}, \ldots, K_n, A']$ iff

$(c_1, \ldots, c_m, a) \in R[K_1, \ldots, K_m, A]$.

else /*A *is a one-to-one or one-to-many attribute, i.e., A and the meta-attributes $C_{m+1}, \ldots, C_n$ together determine the identifier of R. A should be modelled as the identifier of an entity type to preserve the cardinality constraint.*/

Construct $E_{A'}$ with the identifier $A' = A_{ont}$.

Construct a relationship type R' connecting the entity types $E_1, \ldots, E_n$ and $E_{A'}$, such that

$(c_1, \ldots, c_m, c_{m+1}, \ldots, c_n, a) \in R'[K_1, \ldots, K_m, K_{m+1}, \ldots, K_n, A']$ iff

$(c_1, \ldots, c_m, a) \in R[K_1, \ldots, K_m, A]$.

Represent a FD $K_{m+1}, \ldots, K_n, A' \rightarrow K_R$ as the cardinality constraint on R'.

If A is a one-to-one attribute, also represent a FD $K_{m+1}, \ldots, K_n, K_R \rightarrow A'$ on R'.

Step 2 Merge equivalent entity types, relationship types and attributes respectively, and unite their domains. □

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*, chapter 8, 10, pages 173–187, 216–235. Addison-Wesley, 1995.

[2] R. Agrawal, A. Somani, and Y. R. Xu. Storing and querying of e-commerce data. In *VLDB*, pages 149–158, 2001.

[3] C. Batini and M. Lenzerini. A methodology for data schema integration in the

entity-relationship model. *IEEE Trans. on Software Engineering*, 10(6), 1984.

[4] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for schema integration. *CN computing surveys*, 18(4):323–364, 1986.

[5] A. Elmagarmid, M. Rusinkiewicz, and A. Sheth. *Management of heterogeneous and autonomous database systems*. Morgan Kaufmann, 1999.

[6] C. H. Goh, S. Bressan, S. Madnick, and M. Siegel. Context interchange: new features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems*, 17(3), 1999.

[7] G. Gottlob. Computing covers for embedded functional dependencies. In *SIGMOD*, 1987.

[8] Qi He and Tok Wang Ling. Extending and inferring functional dependencies in schema transformation. In *CIKM*, 2004.

[9] C. N. Hsu and C. A. Knoblock. Semantic query optimization for query plans of heterogeneous multidatabase systems. *TKDE*, 12(6):959–978, 2000.

[10] V. Kashyap and A. Sheth. Semantic and schematic similarity between database objects: a context-based approach. *The VLDB Journal*, 5, 1996.

[11] R. Krishnamurthy, W Litwen, and W. Kent. Language features for interoperability of databases with schematic discrepancies. In *SIGMOD*, pages 40–49, 1991.

[12] L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. On efficiently implementing SchemaSQL on a SQL database system. In *VLDB*, pages 471–482, 1999.

[13] L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. SchemaSQL-an extension to SQL for multidatabase interoperability. *TODS*, 2001.

[14] Mong Li Lee and Tok Wang Ling. Resolving constraint conflicts in the integration of ER schemas. In *International Conference on Conceptual Modeling (ER)*, pages 394–407, 1997.

[15] Mong Li Lee and Tok Wang Ling. A methodology for structural conflicts resolution in the integration of entity-relationship schemas. *Knowledge and Information Sys.*, 5:225–247, 2003.

[16] Tok Wang Ling and Mong Li Lee. Issues in an entity-relationship based federated database system. In *CODAS*, pages 60–69, 1996.

[17] Nelson Mendonça Mattos. Integrating information for on demand computing. In *VLDB*, 2003.

[18] R. J. Miller. Using schematically heterogeneous structures. In *SIGMOD*, pages 189–200, 1998.

[19] M. P. Reddy, B. E. Prasad, and A. Gupta. Formulating global integrity constraints during derivation of global schema. *Data & Knowledge Engineering*, pages 241–268, 1995.

[20] E. Sciore, M. Siegel, and A. Rosenthal. Using semantic values to facilitate interoperability among heterogeneous information systems. *TODS*, 19(2), 1994.

[21] A. P. Sheth and S. K. Gala. Federated database systems for managing distributed, heterogenous, and autonomous databases. *ACM Computing Surveys*, 1990.

[22] S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of heterogeneous schemas. *VLDB*, 1992.

[23] M. W. W. Vermeer and P. M. G. Apers. The role of integrity constraints in database interoperation. In *VLDB*, 1996.