Extending and Inferring Functional Dependencies in Schema Transformation

Qi He
heqi@comp.nus.edu.sg

Tok Wang Ling
lingtw@comp.nus.edu.sg

Dept. of Computer Science, School of Computing,
National University of Singapore

ABSTRACT

We study the representation, derivation and utilization of a special kind of constraints in multidatabase systems. A major challenge is when component database schemas are *schematic discrepant* from each other, i.e., data values of one database correspond to schema labels of another. We propose "qualified functional dependencies" (or qualified FDs), an extension to conventional FDs to formalize integrity constraints in multidatabase systems. We first give inference rules to derive qualified FDs in fixed schemas, then study the derivation of qualified FDs during the transformations between schematic discrepant schemas. Propagation rules are given to derive qualified FDs of transformed schemas from qualified FDs of original schemas. Our work can be used in different stages of building and accessing a multidatabase system, e.g., to detect and resolve value inconsistency in schema integration, to verify lossless schema transformations, to normalize integrated schemas, to verify the integrity of data, and to optimize queries at an integration level. In particular, as an application of our theory, we will use FDs to check the validity of SchemaSQL views (SchemaSQL is a powerful multidatabase language).

Categories and Subject Descriptors

H.2.0 [**Database Management**]: General - *Integrity*

General Terms

Theory

Keywords

Functional dependency, Schematic discrepancy, Schema integration, Multidatabase.

1. INTRODUCTION

Schema integration [3, 4, 14] is the activity to integrate 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* [18, 22] (also called *multidatabase system* or *federated database system*), to integrate data in a data warehouse, or to integrate user views in database design. *Schema transformation* is the process to transform heterogeneous schemas into unified ones (In this paper, we'll blur the difference on "integration" and "transformation", and treat schema integration as a special kind of schema transformation). Existing works focused on presenting an integrated view of data available at component schemas. People have developed some methods to resolve naming conflicts (i.e., homonyms or synonyms), structural conflicts (using different schema constructs to model the same concept), and schematic discrepancy in schema integration. A less studied area is on the constraint issues, i.e., how to describe, derive and utilize constraints in a multidatabase environment. In a (individual or heterogeneous) database system, constraints should be enforced to ensure the integrity of data in the operations of insertion, deletion and update. Furthermore, constraints provide semantics which could be used to optimize queries, or to detect redundancy and data inconsistency.

Example 1.1: Suppose we want to integrate two bookstore databases with the same schema: *BS1(isbn, title, price)* and *BS2(isbn, title, price)*. Can we just integrate them into a schema as *BS1* or *BS2*? The answer would be negative if we have the constraint: *a book with an isbn number has the same title but not necessary the same price in the two bookstores*, as value inconsistency would occur on the *price* attribute. Actually, the FD $isbn \rightarrow title$ is a "global" FD holding in the union of the two relations *BS1* and *BS2*, while the FD $isbn \rightarrow price$ only holds in individual relations. It would be better to distinguish a book's prices of different bookstores in the integrated schema, e.g., *Book(isbn, title, BS1_price, BS2_price)* or *Book(isbn, title, store, price)*. □

In individual databases, the issue of inferring view dependencies has been introduced in [1, 7]. However, the representation and derivation of constraints in a multidatabase system would be harder than in an individual database system, because a multidatabase system is usually distributed (i.e., data may be divided and stored in several databases) and heterogeneous (i.e., the similar data may be represented in quite different forms in component databases). In particular, the integrated schema of a multidatabase system is generated by not only relational algebra, but also some other restructuring operators as we will introduce later. And therefore, to derive dependencies for an integrated schema, the existing inference rules for relational algebra are not enough. We also need find rules for those additional restructuring operators.

CIKM'04, November 8–13, 2004, Washington, DC, USA.

In this paper, we'll study the representation, derivation and utilization of an important kind of constraints, FDs, in multidatabase systems. To meet the demand of expressivity of constraints in a multidatabase environment, we will propose qualified FDs which are FDs holding over a set of relations or sub-relations. Inference rules will be designed to derive unknown qualified FDs from known ones in fixed schemas. We will also study the propagation of qualified FDs in schema transformation. A major challenge is when the schemas are schematic discrepant from each other. The interplay of data and schema labels causes schema transformations quite different from classical operations such as union, join, etc. We will explore some propagation rules of qualified FDs in transformations between schematic discrepant schemas.

In the rest of this section, Section 1.1 shows schematic discrepancy and schema transformation by example. Section 1.2 elaborates on some applications of FDs in multidatabase systems, in order to bring out the motivation for our work. Section 1.3 describes the organization of the paper.

1.1 Schematic Discrepancy and Schema Transformation

In relational databases, *schematic discrepancy* occurs when the same information is modeled as attribute values, relation names or attribute names in different databases. Schematic discrepancy arises frequently since names for schema constructs often capture some intuitive semantic information. Some scholars argue that even within the relational model it is more the rule than the exception to find data represented in schema constructs [11]. Recently, people have studied how to query data from discrepant databases [11, 12, 13] and how to use schematic discrepancy [2, 19]. We hereby give an example of schematic discrepancy which will be used as a running example in this paper.

Example 1.2: In Figure 1, we give three databases recording the same information: *supplying prices of products (identified by p#) by suppliers (identified by s#) in different months*. However, the months are modelled respectively as attribute values, attribute names and relation names in *DB1*, *DB2* and *DB3*. For example, in *DB2*, the months *Jan, ..., Dec* are attribute names whose values are prices in those months; in *DB3*, each relation with a month as its name records the supplying information in that month.

In each database, we assume a product's price is functionally dependent on the product number, the supplier number and the month. This constraint is expressed as different FDs in the three databases.□

To integrate or interoperate with schematic discrepant databases, people need to transform discrepant schemas into unified forms. We call a transformation between schematic discrepant schemas a ***schematic discrepant transformation***. In [12], Lakshmanan et al developed four restructuring operators, *fold, unfold, unite* and *split* (originally introduced in the context of the *tabular algebra* [8]), to implement such transformations. In what follow, we first give the formal definition of these operators, then explain them by Figure 1.

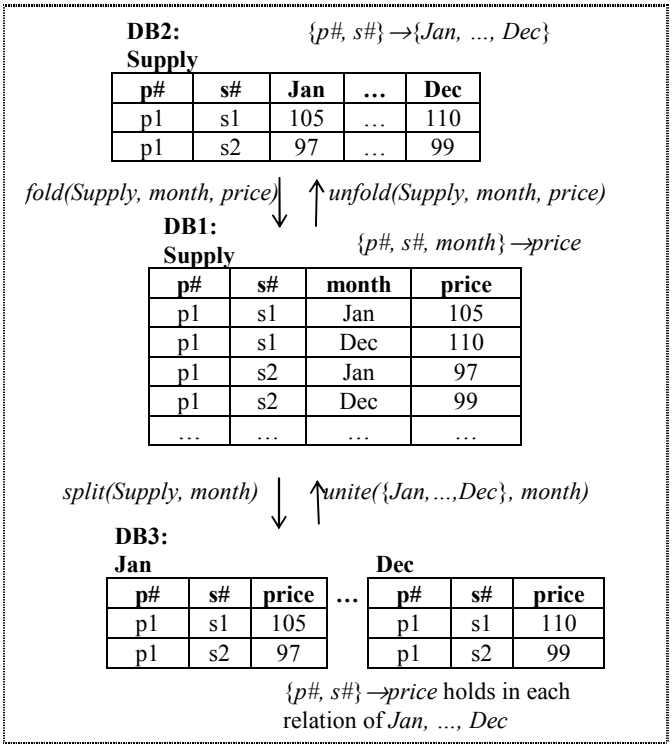


Figure 1: Schematic discrepancy: months modeled differently in *DB1~DB3*

***unfold*(R, B, C)**: Let R be a relation with the schema $R(A_1, ..., A_n, B, C)$. *unfold(R, B, C)* transforms R to a relation $S(A_1, ..., A_n, b_1, ..., b_m)$, where $\{b_1, ..., b_m\}$ is the set of distinct values appearing in column B of R. The content of S is defined as:

$$S=\{(a_1, ..., a_n, c_1, ..., c_m) \mid (a_1, ..., a_n, b_i, c_i) \in R, 1 \le i \le m\}.$$

***fold*(R, B, C)**: Let R be a relation with the schema $R(A_1, ..., A_n, b_1, ..., b_m)$. Suppose $b_1,...,b_m$ are values from $dom(B)$, and all entries appearing in columns $b_1, ..., b_m$ of R are from $dom(C)$, for some attribute names $B,C \notin \{A_1, ..., A_n\}$. *fold(R, B, C)* transforms R to a relation $S(A_1, ..., A_n, B, C)$, defined as:

$$S = \{(a_1, ..., a_n, b_i, c_i) \mid \exists\, t \in R: t[A_1,...,A_n]= (a_1,...,a_n) \,\&\, t[b_i]=c_i\}.$$

***split*(R, B)**: Let R be a relation with the schema $R(A_1, ...,A_n, B)$. *split(R, B)* transforms R to a set of relations $b_i(A_1, ..., A_n)$, for each b_i appearing in column B of R. The content of b_i is defined as:

$$b_i = \{t[A1,...,An] \mid t \in R \,\&\, t[B]= b_i\}.$$

***unite*($\mathcal{R}_B$, B)**: Let $\mathcal{R}_B=\{b_1,...,b_m\}$ be a set of relations in a given database, such that each relation name b_i (i=1,2,…,m) is an element of the domain of some fixed attribute B, and all the relations have a common schema $b_i(A_1,...,A_n)$. *unite*($\mathcal{R}_B$, B) transforms the relations $b_1,...,b_m$ to a relation $S(B, A_1,...,A_n)$, defined as:

$$S = \{t \mid \exists\, t' \in b_i: t[A1,...,An] = t'[A1,...,An] \,\&\, t[B]= b_i\}.$$

For example, in Figure 1, these operators are used to implement transformations between those discrepant databases, as described below:

- The *unfold* operator transforms the *Supply* relation of *DB1* into the relation of *DB2*. It takes the original relation name and two attribute names as its parameters.
- The *fold* operation with the similar parameters is a converse transformation of *unfold*. Note the original relation name *Supply* refers to the relation of *DB2* now. We suppose we know in advance that the attribute names *Jan, …, Dec* in the original relation of *DB2* are values of some attribute *month*, and their values are from the domain of another attribute *price*.
- The *split* operator transforms the relation of *DB1* into the set of relations of *DB3*. It takes the original relation name and an attribute name as its parameters.
- The *unite* operation with the set of original relation names and an attribute as its parameters is a converse transformation of *split*. Again, we know in advance that the relation names *Jan, …, Dec* of *DB3* are values of some attribute *month*.

To focus our work, in this paper, we study the derivation of qualified FDs in a schematic discrepant transformation that is a sequence of restructuring operators. In the future, we'll extend our work to more general transformations which include both restructuring operators and relational algebra (union, natural join, etc).

1.2 Applications of FDs in Multidatabase Systems

As a special kind of integrity constraints, FDs play important roles in relational databases. Although the inference and applications of FDs in individual database systems have been studied for decades, the same issue in multidatabase systems is less studied. It is not trivial to derive the (qualified) FDs on an integrated schema from the (qualified) FDs on component schemas, especially in the presence of schematic discrepancies, as we'll show in this paper. The issue is interesting as FDs are useful not only in enforcing the integrity of data, but also in different stages of building and accessing a multidatabase system. In what follows, we'll identify some applications of FDs in schema transformation, schema normalization and query processing. A more special application of our theoretical work in multidatabase interoperability will be introduced in Section 4 below.

- *Verifying Lossless Transformations.* As mentioned, schema transformation plays an important role in building a multidatabase system. In practice, one is mostly interested in semantic-preserving transformations. A set of relations can be losslessly converted into another set of relations, and conversely, hence the name of *lossless transformation.* In other words, a lossless transformation defines a one to one mapping from the instance set of the original relations onto the instance set of the transformed relations. FDs can be used to verify not only "lossless join decomposition", but also lossless schematic discrepant transformation, as shown in the following example.

Example 1.3: Given a relation with the schema *R(A, b1, b2)*, suppose the attribute names *b1, b2* are values of a fixed attribute *B*, and the values of the attributes *b1*, *b2* (i.e., *c1, c2*, etc) are from the domain of another attribute *C*. By applying the operation *fold(R, B, C)*, we can transform either of the two instances I1, I2 of *R* (Figure 2) into the same relation instance of *S*. That is, the mapping from the instances of *R* onto the instances of *S* is many to one, which makes the recovering impossible. So the *fold* operation is lossy (non-lossless). On the other hand, if the FD $A \rightarrow \{b1, b2\}$ held in *R*, the transformation would be lossless. □

R (I1)

A	b1	b2
a1	c1	c2
a1	c3	c4

R (I2)

A	b1	b2
a1	c1	c4
a1	c3	c2

⇓ ⇓

S

A	B	C
a1	b1	c1
a1	b2	c2
a1	b1	c3
a1	b2	c4

Figure 2: A "lossy" (non-lossless) *fold* transformation

- *Normalizing Integrated Schemas.* Consolidating data into a single physical store has been the most effective approach to provide fast, highly available, and integrated access to related information. The applications include coalescing all the required data for a new e-business application for online transactions, and enabling sophisticated data mining of warehoused historical data. In classical relational theory, FDs are used to detect redundancy and normalize relations. Deriving FDs for integrated schemas becomes important, as schematic discrepant transformation would introduce redundancy.

For example, in Example 1.1, the integrated schema *Book(isbn, title, store, price)* is not in 2nd normal form, as the FDs $isbn \rightarrow title$ and $\{isbn, store\} \rightarrow price$ hold in the relation (the method to derive the two FDs will be introduced in Section 3). We can normalize it to two relations: *Book(isbn, title)* and *BookPrice(isbn, store, price)*.

- *Semantic Query Optimization.* In a multidatabase system, FDs and other constraints on integrated schemas could be used to optimize queries against the integrated view [10, 16], to eliminate subqueries which are known to yield empty results, and to validate update transactions at the integration level [21].

For example, in Example 1.1, given the integrated schema *Book(isbn, title, store, price)*, a query retrieves books with the same isbn number but different titles in the two bookstores. Such a query will return empty results as the FD $isbn \rightarrow title$ holds in the integrated *Book* relation.

1.3 Paper Overview

The main contributions of this paper are in Section 2 ~ 4. In Section 2, we extend conventional FDs to *qualified FDs*, to express FD-like constraints in multidatabase systems. Inference rules of qualified FDs are also given in this section. Then in Section 3, we study the propagation of qualified FDs in schematic discrepant transformations. In Section 4, we show a special application of our theoretical work in multidatabase interoperability, i.e., use FDs to check the validity of SchemaSQL views. In Section 5, we compare our work with some related work. Section 6 is for the conclusion and future work, in which we introduce some preliminary study on the inference and application of FDs on XML data.

Notations: In this paper, capital letters near the beginning of the alphabet, *A, B, C,* stand for single attributes. Capital letters near the end of the alphabet, *U, X, Y, Z*, stand for sets of attributes. Lower case letters stand for attribute values, which may be modeled as attribute or relation names in discrepant schemas however. The notation *dom(A)* stands for the domain of an attribute *A*.

2. QUALIFIED FD

In this section, we introduce *qualified FD*, an extension to conventional FD, to facilitate the expression and inference of some constraints in multidatabase systems. We first give the formal definition of qualified FDs in Section 2.1. Then in Section 2.2, we'll give some inference rules of qualified FDs in fixed schemas.

2.1 Definition of Qualified FD

We first give an example to show the motivation of our proposal.

Example 2.1: Suppose in *DB3* of Figure 1,

Jan			…	Dec		
p#	s#	price		p#	s#	price

the following constraint holds: *in the first quater, each product is supplied with the same price by suppliers s1 and s2, independent of months.* We'll express this constraint as:

$$\{Jan, Feb, Mar\}(s\#_{\sigma=\{s1, s2\}}, p\# \rightarrow price).$$

Jan, Feb and *Mar* are relation names from *DB3*. The beginning part "{*Jan, Feb, Mar*}" restricts the set of relations, and "$s\#_{\sigma=\{s1,s2\}}$" restricts the values of the attribute *s#* within which the constraint should be enforced. Formally, let

$$R = \cup_{mi \in \{Jan, Feb, Mar\}}(\sigma_{s\# \in \{s1, s2\}} mi)$$

the FD $p\# \rightarrow price$ holds in *R*.□

The definition below formalizes the above representation of qualified FDs.

Definition 2.1 (*Qualified FD*): In general, given a set of relations $\mathcal{S}$ with the same set of attributes *U*, we can represent a qualified FD as:

$$\mathcal{R}(A_1{}_{\sigma=S_1}, A_2{}_{\sigma=S_2}, ..., A_n{}_{\sigma=S_n}, X \rightarrow Y).$$

Syntax of the qualified FD:

(1) $\mathcal{R} \subseteq \mathcal{S}$ represents the set of relations over which the qualified FD holds.

(2) $A_i{}_{\sigma=S_i}$, for each $i=1, ..., n$, satisfies $A_i \in U$ and $S_i \subseteq dom(A_i)$, indicating the restriction of attribute values within which the qualified FD holds. For easy to reference, we call each $A_i{}_{\sigma=S_i}$ a *qualification attribute* from *U*.

(3) $X \subseteq U$ and $Y \subseteq U$ are two sets of *regular attributes*.

Semantics of the qualified FD:

We call the given qualified FD holds over $\mathcal{R}$, if the following holds for any two tuples *t1*, *t2* from the relations of $\mathcal{R}$ (*t1, t2* may come from one or two relations):

If $t1.A_i \in S_i$ and $t2.A_i \in S_i$, for each $i = 1, ..., n$, and $t1.X_j = t2.X_j$, for each attribute $X_j \in X$, then $t1.Y_k = t2.Y_k$, for each attribute $Y_k \in Y$. This completes the definition of qualified FD. □

In general, if a qualified FD

$$\{R_1, ..., R_m\}(A_1{}_{\sigma=S_1}, ..., A_n{}_{\sigma=S_n}, X \rightarrow Y)$$

holds, let $R = \cup_{i=1,...,m}(\sigma_{A_1 \in S_1, ..., A_n \in S_n} R_i)$, then the FD $X \rightarrow Y$ holds in *R*. If a qualified FD only contains regular attributes and holds in a single relation, then it is just a conventional FD.

2.2 Reasoning about Qualified FDs in Fixed Schemas

In general, let *F* be a set of qualified FDs for a set of relation schemas $\mathcal{R}$, and let *f* be a qualified FD also for $\mathcal{R}$. We say *F logically implies f*, if every instance of $\mathcal{R}$ that satisfies the dependencies in *F* also satisfies *f*. We define F^+, the ***closure of F*** for $\mathcal{R}$, to be the set of qualified FDs for $\mathcal{R}$ that are logically implied by *F*.

To understand logical implications among qualified FDs in fixed schemas, we provide a complete set of inference rules, meaning that from a given set of qualified FDs *F* for $\mathcal{R}$, the rules allow us to deduce all the true qualified FDs for $\mathcal{R}$, i.e., those in F^+. Without causing confusion, in the next section, we'll give another kind of rules (called *propagation rules*) which allow us to infer qualified FDs of transformed relations from qualified FDs of original relations in a schema transformation.

Some of the inference rules are given below (a complete set of inference rules are given in Appendix A). We assume for each qualification attribute $A_{\sigma=S}$, the domain of *A* is a finite and fixed set.

Inference rules of qualified FDs: Given a set of relation schemas $\mathcal{S}$ with the same set of attributes *U*, and a set of qualified FDs *F* for $\mathcal{S}$, let $\overline{X}$ be a mixed set of regular and qualification attributes from *U* ($\overline{X}$ may comprise only regular or qualification attributes) [1]; let $Y \subseteq U$ and $Z \subseteq U$ be two sets of regular attributes; let attribute $A \in U$; let $\mathcal{R}1 \subseteq \mathcal{R} \subseteq \mathcal{S}$, $S1 \subseteq S \subseteq dom(A)$, we have the following inference rules:

(A1) *Partition on the relation set*. If $\mathcal{R}(\overline{X} \rightarrow Y)$ holds, then $\mathcal{R}1(\overline{X} \rightarrow Y)$ holds.

(A2) *Partition on the qualification.* If $\mathcal{R}(A_{\sigma=S}, \overline{X} \rightarrow Y)$ holds, then $\mathcal{R}(A_{\sigma=S1}, \overline{X} \rightarrow Y)$ holds.

(A3) *Disassembly.* $\mathcal{R}(\overline{X}, A \rightarrow Y)$ holds iff $\mathcal{R}(A_{\sigma=\{a\}}, \overline{X} \rightarrow Y)$ holds for each $a \in dom(A)$.

(A4) *Reflexivity.* If $Y \subseteq \overline{X}$, then $\mathcal{R}(\overline{X} \rightarrow Y)$ holds.

(A5) *Augmentation.* If $\mathcal{R}(\overline{X} \rightarrow Y)$ holds, then $\mathcal{R}(\overline{X}, Z \rightarrow Y, Z)$ holds.

(A6) *Transitivity.* If $\mathcal{R}(\overline{X} \rightarrow Y)$ and $\mathcal{R}(\overline{X1}, Y \rightarrow Z)$ hold, where $\overline{X1}$ is a set (possibly an empty set) of some qualification attributes of $\overline{X}$, then $\mathcal{R}(\overline{X} \rightarrow Z)$ holds. □

Rule A1 and A2 are trivial. Rule A3 is a derived rule from the rules in Appendix A, which is useful in the rest of the paper. We hereby explain this rule through an example:

Example 2.2: In *DB1* of Figure 1, given the FD

$$\{month, p\#, s\#\} \rightarrow price$$

[1] This convention will be followed in the rest of the paper.

we can infer a set of qualified FDs in the same relation by the disassembly rule, i.e.,

$$Supply(month_{\sigma=\{mi\}}, p\#, s\# \rightarrow price)$$

for each $mi \in \{Jan, ..., Dec\}$. That is, the FD $\{p\#, s\#\} \rightarrow price$ holds in each sub-relation $\sigma_{month=mi}$ *Supply* of *DB1*. □

Rules A4, A5 and A6 extend Armstrong's Axioms [20], the inference rules of FDs. Note in Rule A6, the inferred qualified FD inherits all the qualification attributes of the given qualified FDs. A sound and complete set of inference rules is given in Appendix A, stated as follows:

Theorem 2.1: The inference rules in Appendix A are sound and complete. □

We can prove this by showing that if F is the given set of qualified FDs holding in $\mathcal{S}$, and f is a qualified FD which cannot be proved by the inference rules, then there must be an instance of $\mathcal{S}$ in which the dependencies of F all hold but f does not; that is, F does not logically imply f. For the detailed proof and more information on qualified FDs (e.g., the computation of attribute closure with respect to a set of qualified FDs, and the implication algorithm), please refer to the full paper [9].

3. PROPAGATION OF QUALIFIED FDS IN SCHEMA TRANSFORMATION

In this section, the implication of qualified FDs extends to transforming schemas. In general, given a schema transformation T, let $\mathcal{R}$ and $\mathcal{S}$ be the sets of original and transformed relations of T; let F be a set of qualified FDs for the schemas of $\mathcal{R}$, and f be a qualified FD for the schemas of $\mathcal{S}$; let r be any instance of $\mathcal{R}$ satisfying the dependencies of F, and s be the instance of $\mathcal{S}$ transformed from r by T. We say F (*logically*) *implies* f, if s satisfies f.

Note unlike the implication of qualified FDs in fixed schemas, now the given set of dependencies F and the implied one f hold in different schemas. To understand logical implications among qualified FDs in transforming schemas, we provide a set of propagation rules, meaning that from a given set of qualified FDs F for the set of original relations $\mathcal{R}$, the rules allow us to deduce the qualified FDs for the set of transformed relations $\mathcal{S}$.

In this section, we first give the *propagation rules* for *split/unite* and *unfold/fold* operations in a pairwise way, by which we can compute qualified FDs on transformed relations from qualified FDs on original relations in application of those operators. Then we propose a method to infer qualified FDs in a schematic discrepant transformation (i.e., a sequence of restructuring operators) using the inference rules (Appendix A) and propagation rules.

3.1 Propagation Rules

We first give the propagation rules for *split/unite* operators then for *unfold/fold*. The soundness of these rules are proven in [9].

Propagation of qualified FDs in application of a split/unite operator: Let $R(A_1, ..., A_n, B)$ be an original relation with $dom(B) = \{b_1, ..., b_m\}$, and $b_i(A_1, ..., A_n)$, $i = 1, ..., m$, be the transformed relations using *split(R, B)*, i.e., the distinct values of B in R, $\{b_1, ..., b_m\}$, become the relation names of the transformed relations. Let $\overline{X}$ be a mixed set of regular and qualification attributes from $\{A_1, ..., A_n\}$, and $Y \subseteq \{A_1, ..., A_n\}$ be a set of regular attributes; let $\mathcal{R}_B \subseteq \{b_1, ..., b_m\}$ be a set of relation names. We have the following rule:

(P1) $R(B_{\sigma=\mathcal{R}_B}, \overline{X} \rightarrow Y)$ holds iff $\mathcal{R}_B(\overline{X} \rightarrow Y)$ holds.

The same rule holds for the *unite* operator, when $\{b_1, ..., b_m\}$ are the original relations, and R is the transformed relation using $unite(\{b_1, ..., b_m\}, B)$.□

Rule P1 means that in application of a *split* operator, the restriction on the values of attribute B in the given qualified FD becomes the restriction on the relation set over which the inferred qualified FD holds, as B values become relation names in the transformed schemas. We hereby give an example to apply this rule:

Example 3.1: In Figure 1, given the FD in the relation *Supply* of *DB1*: $\{p\#, s\#, month\} \rightarrow price$ which is equivalent to a set of qualified FDs in the same relation (by the disassembly rule A3): $Supply(month_{\sigma=\{mi\}}, p\#, s\# \rightarrow price)$ for each $mi \in \{Jan, ..., Dec\}$, we can derive a FD for each relation of *DB3* by applying the propagation rule P1 to each of the qualified FDs in *DB1*, i.e.,

$$mi(p\#, s\# \rightarrow price)$$

for each relation name $mi \in \{Jan, ..., Dec\}$ in *DB3*.□

Although *unite* is a qualified-FD preserving transformation, *split* is not. Given the same conditions as those in Rule P1, a qualified FD $R(\overline{X} \rightarrow B)$ will be lost in application of *split*, as the values of B become relation names in the transformed schemas.

In what follows, we'll give the propagation rules of qualified FDs in application of a set of *unfold/fold* operators. We study based on a set of *unfold/fold* operators instead of individual ones because some qualified FDs would hold over a set of relations (which are transformed together by *unfold/fold* operations).

Propagation of qualified FDs in application of a set of unfold/fold operators: Let $R_i(A_1, ..., A_n, B, C)$, $i = 1, ..., l$, be a set of original relations, and $S_i(A_1, ..., A_n, b_1, ..., b_m)$, $i=1, ..., l$, be the set of transformed relations by performing $unfold(R_i, B, C)$ on each relation of R_i. That is, the values of B in R_i, $\{b_1, ..., b_m\}$, become attribute names in S_i, and the values of C in R_i become the values of the attributes $b_1, ..., b_m$ in S_i. Let $\overline{X}$ be a mixed set of regular and qualification attributes from $\{A_1, ..., A_n\}$, and $Y \subseteq \{A_1, ..., A_n\}$ be a set of regular attributes. Let $\mathcal{R}=\{R_{i1}, ..., R_{ij}\}$ be a subset of $\{R_1, ..., R_l\}$, and $\mathcal{S}=\{S_{i1}, ..., S_{ij}\}$, a subset of $\{S_1, ..., S_l\}$, be the transformed relations from $\mathcal{R}$. We have the following rules:

(P2) $\mathcal{R}(B_{\sigma=\{b_i\}}, \overline{X} \rightarrow C)$ holds iff $\mathcal{S}(\overline{X} \rightarrow b_i)$ holds.

(P3) $\mathcal{R}(B_{\sigma=\{b_i\}}, \overline{X}, C \rightarrow Y)$ holds iff $\mathcal{S}(\overline{X}, b_i \rightarrow Y)$ holds.

(P4) $\mathcal{R}(\overline{X} \rightarrow Y)$ holds iff $\mathcal{S}(\overline{X} \rightarrow Y)$ holds.

The three rules also hold for *fold* operators, when S_i, $i = 1, ..., l$, are the original relations, and R_i, $i=1, ..., l$, are the transformed relations by performing $fold(S_i, B, C)$ on each relation of S_i. □

In application of *unfold* operators, Rule P2 and P3 mean that the restriction on the value of attribute B in the given qualified FD becomes the restriction on the attribute name in the inferred qualified FD. Rule P4 is trivial as no change happens on the attributes involved in the given qualified FD during the transformation. Note both *fold* and *unfold* operations are not

qualified-FD preserving transformations. We hereby give an example to apply Rule P2:

Example 3.2: In Figure 1, given the FD in the relation *Supply* of *DB1*: $\{p\#, s\#, month\} \rightarrow price$ which is equivalent to a set of qualified FDs in the same relation: $Supply(month_{\sigma=\{mi\}}, p\#, s\# \rightarrow price)$ for each $mi \in \{Jan, ..., Dec\}$, we can derive a set of FDs in *DB2* by applying Rule P2 on each of the qualified FDs in *DB1*, i.e.,

$$Supply(p\#, s\# \rightarrow mi)$$

for each attribute name $mi \in \{Jan, ..., Dec\}$. That is, the FD $\{p\#, s\#\} \rightarrow \{Jan, ..., Dec\}$ holds in relation *Supply* of *DB2*. □

3.2 Inferring Qualified FDs in Schematic Discrepant Transformation

Using the inference rules in Appendix A and the propagation rules P1~P4, we can design an algorithm to derive qualified FDs in schematic discrepant transformations. A naive idea would be: for each step of a schematic discrepant transformation, we first apply the inference rules to compute the qualified FD closure on the original relations, then apply the propagation rules to get the qualified FDs on the transformed relations. However, the computation of qualified FD closures takes exponential time at least, which makes the method impractical. Instead of applying the inference and propagation rules directly, we use some derived rules (see Appendix B) to infer qualified FDs in schema transformation, without computing qualified FD closures. The basic idea of the derived rules is: given a set (not necessary a closure) of qualified FDs F on the original relations, we propagate not only the dependencies in F, but also those which are not in but implied by F, and can be preserved during the schema transformation. The general algorithm with proofs is given in [9]. We hereby present an example to explain it.

Example 3.3: Suppose in Figure 1, we have another database *DB4* with supplier numbers as the relation names:

DB4:

s1

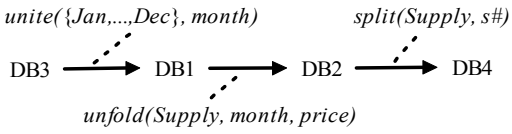
p#	Jan	…	Dec

…

sn

p#	Jan	…	Dec

Suppose we transform *DB3* into *DB4* using the following sequence of restructuring operators:



Given a set of qualified FDs in the relations of *DB3*:

$$mi(p\#, s\# \rightarrow price)$$

for each relation name $mi \in \{Jan, ..., Dec\}$, we compute the qualified FDs in *DB4* as follows:

After applying the *unite* operator, we get the dependencies in *DB1* (by Rule P1):

$$Supply(month_{\sigma=\{mi\}}, p\#, s\# \rightarrow price)$$

for each $mi \in \{Jan, ..., Dec\}$. After applying the *unfold* operator, we get the dependencies in *DB2* (by Rule P2):

$$Supply(p\#, s\# \rightarrow mi)$$

for each $mi \in \{Jan, ..., Dec\}$. These FDs cannot be transformed into any dependencies on *DB4* by the propagation rules. However, each of the above FDs implies a set of qualified FDs on the same relation of *DB2* (by Rule A3):

$$Supply(s\#_{\sigma=\{sj\}}, p\# \rightarrow mi)$$

for each $sj \in \{s1, ..., sn\}$. Consequently, after the *split* operation, these qualified FDs become FDs in *DB4* (by Rule P1):

$$sj(p\# \rightarrow mi)$$

for each relation name $sj \in \{s1, ..., sn\}$ and attribute name $mi \in \{Jan, ..., Dec\}$. That is, in each relation *sj* of *DB4*, the FD $p\# \rightarrow \{Jan, ..., Dec\}$ holds.□

Before ending this section, we show some results on the completeness of our method. The inference rules in Appendix A and the propagation rules P1~P4 are not complete to infer all the qualified FDs in schematic discrepant transformation. Instead, they are complete to infer common dependencies satisfying three conditions:

Definition 3.1: Let $\mathcal{S}$ be a set of relations with the same set of attributes U in database *DB*. We call a qualified FD f: $\mathcal{R}(X \rightarrow Y)$ ***reasonable*** if it satisfies 3 conditions:

(1) Either $\mathcal{R} = dom(B)$ for some attribute B whose values are modeled as relation names in $\mathcal{S}$, or $\mathcal{R}$ is a single relation name from $\mathcal{S}$.
(2) The qualified FD only has regular attributes.
(3) For each attribute set $Z = \{b_i \mid b_i \in U$ is a value of an attribute B, and the values of b_i are from the domain of another attribute $C\}$, there's at most one attribute of Z in $X \cup Y$. □

Without giving proof here (the proof is given in [9]), we have the following results on the completeness of our method using the propagation rules or derived rules:

Theorem 3.1: The inference rules in Appendix A and the propagation rules P1~P4 are complete to infer reasonable qualified FDs in schematic discrepant transformation. □

Proposition 3.1: The derived rules in Appendix B are complete to infer reasonable qualified FDs in schematic discrepant transformation. □

4. VERIFYING SCHEMASQL VIEWS

In this section, we will show an application of our theory in a multidatabase query language SchemaSQL [12, 13]. SchemaSQL is an extension to SQL for enabling multidatabase interoperability. It treats data and schema labels in a uniform manner, i.e., variables can range over data and schema labels, which facilitates the interoperability among schematic discrepant databases. Recently, SchemaSQL has been used to solve a broad range of problems [12, 19]. However, a SchemaSQL view definition may generate ambiguous results. We call those problematic views not "well-defined". The problem can be detected using FDs derived during the processing of SchemaSQL views. In this section, we first define "well-defined SchemaSQL view" in Section 4.1, then show how to verify well-defined views by deriving qualified FDs in Section 4.2.

4.1 Well-defined SchemaSQL Views

In this sub-section, we first show an example of problematic SchemaSQL view which generates ambiguous results, then give

the definition of well-defined views. We consider views for query purpose, not for update purpose.

Though a SQL view defines a mapping from the instances of original relations onto the instances of view relations, a SchemaSQL view defines a mapping from original relations onto view relations including schemas and instances both. That is to say, a SchemaSQL view may define on (and generate) relations with variable schemas.

Example 4.1: In Figure 3, suppose in the relation *Supply*, a FD {*product, supplier, month*} → *price* holds. The SchemaSQL statements below define a view *SupView*:

```
create view   SupView(product, T.month)
select              T.product, T.price
from          Supply  T
```

Supply

product	supplier	month	price
p1	s1	Jan	100
p1	s1	Feb	105
p1	s2	Jan	95
p1	s2	Feb	97

Allocated table

product	Jan	Feb
p1	100	-
p1	-	105
p1	95	-
p1	-	97

SupView(I1)

product	Jan	Feb
p1	100	105
p1	95	97

SupView(I2)

product	Jan	Feb
p1	100	97
p1	95	105

Figure 3: Ambiguous SchemaSQL view: *SupView* may have one of two instances I1 and I2

The above statements are similar to a SQL view definition except a variable *T.month* in the "*create view*" clause. The result view schema therefore depends on the instantiation of *T.month*, i.e., the values of the *month* attribute in the *Supply* relation. In this case, the view has a schema of *SupView(product, Jan, Feb)*. To evaluate this view, they will temporarily generate an "allocated table" shown in Figure 3. Each tuple in the allocated table comes from a tuple of the *Supply* relation with the values of *month* modeled as attribute names. "-" is used to denote the null value. Then they merge the tuples in the allocated table, and get the final result. Two tuples are merge-able if for a common attribute, either the attribute values of the 2 tuples are the same, or at least one value is null. For example, the 1st tuple can be merged with the 2nd or 4th tuple. Then the result view relation is not unique for the different choices of merging tuples. Two possible results are *SupView(I1)* and *SupView(I2)* in Figure 3. That is, the mapping from the original relations onto the view relations is one to many. □

We call a view definition in Example 4.1 is not well-defined. In general, we have:

Definition 4.1 ***(Well-defined SchemaSQL view)*:** Let V be a view definition in SchemaSQL. Let $S1$ = {R | R is an original relation (or relation set) on which V is defined}, $S2$ = {R | R is a view relation (or relation set) generated by V}. If the view definition V: $S1 \longrightarrow S2$ is a many to one mapping, we call V is ***well-defined***. □

Intuitively, for a well-defined view V, given a query Q against a view relation (or relation set) $S \in S2$, we have:

$$Q(S) = Q(V(R)) = Q \circ V(R), \text{ for some } R \in S1.$$

That is, the query Q against S is mapped onto the unique query $Q \circ V$ against the original relation (or relation set) R, if V is a many to one mapping.

4.2 Verifying Well-defined SchemaSQL Views Using FDs

In this sub-section, we first give a theorem, then a method to verify well-defined SchemaSQL views. The theorem below gives a necessary and sufficient condition to check whether a SchemaSQL view is well-defined by use of FDs. To simplify the expression, the theorem only applies to SchemaSQL views generating individual relations without aggregations. The result can be extended to general SchemaSQL views readily.

Theorem 4.1: A SchemaSQL view is well-defined iff it satisfies the following condition: if the output schema declaration through the *create view* statement of the view definition has a form of "$R(A_1, \ldots, A_n, B)$", where R is the name of the view relation, $A_1, \ldots, A_n$ are attribute names, and B is a variable ranging over a set of values $\{b_1, \ldots, b_m\}$, then the FDs

$$\{A_1, \ldots, A_n\} \rightarrow b_i, \; i = 1, \ldots, m, \text{ hold in } R. \; \square$$

In general, when the declaration of a view schema contains a variable, the mapping from the original relations onto the view relations is many to many. However, if certain FDs hold, we can ensure the mapping be many to one. The detailed proof of this theorem is given in [9].

Note according to the SchemaSQL syntax [13], there's at most one variable in the attribute list of the output schema declaration through a *create view* statement. And Theorem 4.1 implies that if a view definition does not contain a variable in the attribute list of the output schema declaration, then the view is always well-defined. That is, Theorem 4.1 could be used to check all the SchemaSQL views which generate single relations without aggregations.

According to Theorem 4.1, in order to check whether a SchemaSQL view is well-defined, we need to infer FDs holding in the view relation. SchemaSQL queries/views can be implemented by use of the restructuring operators and relational algebra (selection, projection, join, and so on) [12]. Consequently, we need develop propagation rules and algorithms to infer qualified FDs in application of the relational algebra besides those four restructuring operators, which are omitted here. We hereby give an example to describe this process.

Example 4.2: The view of Example 4.1 can be implemented in two steps: (1) project out the *supplier* column from the *Supply* relation, and get an intermediate relation, say *Sup1(product, month, price)*; (2) perform *unfold(Sup1, month, price)*, and get the result relation *SupView*. As Step (1) projects out the *supplier* attribute, the given FD {*product, supplier, month*} →*price* is lost

after the projection. Consequently, no FD holds in *SupView*, which means the view is not well-defined.

On the other hand, if the view schema declaration contains the attribute *supplier*, i.e., *SupView(product, supplier, Jan, Feb)*, then the view is implemented by performing *unfold(Supply, month, price)*. Using Rule A3 and P2, we can derive a FD {*product, supplier*}→{*Jan, Feb*} in *SupView*. According to Theorem 4.1, the view is well-defined. □

5. RELATED WORK

Most of the existing relational dependencies, such as *FDs, multivalued dependencies, embedded multivalued dependencies* etc, are defined on individual relations. They differ from our proposal of qualified FDs which are constraints on a set of relations or sub-relations. Some unifying frameworks were proposed to generalize those existing dependencies. One of the most powerful methods is to use tableaux (a table form representation) to present constraints, and use "chase" (a procedure based on the successive application of constraints to tableaux) to analyze implication and construct axiomatization [1]. However, the existing tableaux paradigm does not subsume our proposal of qualified FDs which have restrictions on attribute values and hold over the union of a set of relations. And the inference rules developed in the chase paradigm cannot be used to infer qualified FDs.

Another kind of extension to FDs in the database design world are FDs partially holding in a relation, in the sense that only some tuples, called exceptions, break the dependencies. These dependencies include *weak FDs* [17]*, afunctional dependencies* [6] and *partial FDs* [5]. The difference between those dependencies and qualified FDs is that the former ones work over instances while the qualified FDs are defined on schemas. Given a relation schema, a weak FD (or some other similar dependency) predicates that some tuples (but don't know which tuples) in the relation would violate the dependency, while a qualified FD indicates exactly what kind of tuples satisfy the dependency. Furthermore, we are not aware of any axiomatizations for those dependencies. At last, those dependencies are specified on individual relations, while qualified FDs can be on a set of relations.

Some work [15, 21, 23] has been done on the derivation of constraints in schema integration. Those works are based on semantic rich schemas (ER schema or object oriented schema). They failed to consider schematic discrepancy in schema integration; neither did they prove the completeness of their methods.

6. CONCLUSION AND FUTURE WORK

In [16], Chen Li introduced some open problems and preliminary study on describing and utilizing constraints to answer queries in data integration systems. Our work solved some of those problems. In particular, we have made three contributions on the representation, derivation and utilization of constraints in multidatabase systems: (1) We proposed qualified FDs to formalize some constraints in multidatabase systems. We gave a complete set of inference rules to derive qualified FDs in fixed schemas. (2) We gave the propagation rules of qualified FDs in application of the restructuring operators, and proposed a method to derive qualified FDs in schematic discrepant transformations. Our work can be used to verify lossless schema transformations, normalize transformed/integrated schemas, optimize queries at the integration level and so on in building and accessing a multidatabase system. (3) As a special application of our theoretical work in multidatabase interoperability, we showed how to use FDs to check well-defined SchemaSQL views in detail.

Information integration using XML as a standard to represent and exchange data provides a competitive advantage to businesses. However, the flexibility of XML also brings great challenge in the integration of XML data from different sources. Although our work in this paper is based on the relational model, the results could be extended to the hierarchical model of XML as well. We are currently studying this problem. In the example below, we show some ideas on the application and derivation of FDs in the integration of XML schemas.

Example 6.1: In Figure 4, we represent XML schemas as tree structures in which elements are represented as rectangles and attributes as circles (filled circles denote keys of the owning elements). The schemas X1, …, Xn model the book information of n bookstores with the store names *s1, ..., sn*. We assume a book with an isbn number has the same title and authors but not necessary the same price in those bookstores. That is, the FD *isbn→title* holds over the union of the instances of X1, …, Xn. However, the FD *isbn→price* only holds in each Xi.

Guided by these dependencies, we can integrate these schemas by transforming the schema labels *s1, ..., sn* into attribute values of a new created element *store*. The integrated schema is also given in Figure 4. Note the attribute *price* is attached to the element *store* now, as its values depend on both isbn numbers and bookstore names. Actually, *price* is an attribute of the relationship type between *book* and *store*. We can derive the FDs *isbn→title* and {*isbn, s_name*}*→price* in the integrated schema.□

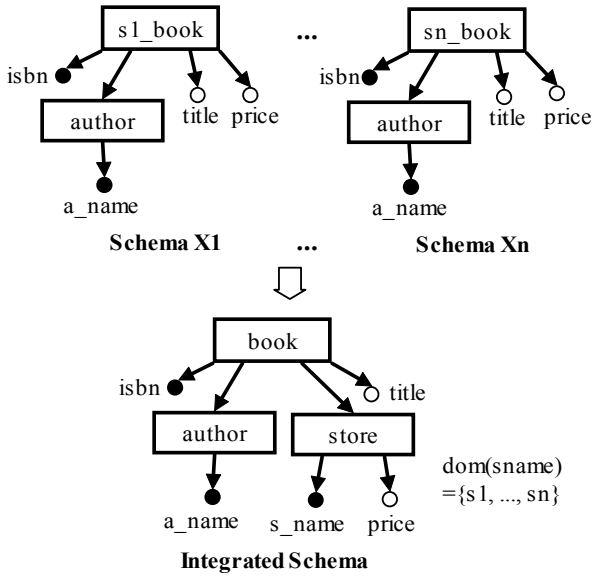


Figure 4: Integration of XML schemas. The values of the attribute *price* depend on isbn numbers and bookstore names.

7. REFERENCES

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

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

[3] J. Albert. Theoretical Foundations of Schema Restructuring in Heterogeneous Multidatabase Systems. *CIKM*, 2000.

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

[5] F. Berzal, J. C. Cubero, F. Cuenca, J. M. Medina. Relational decomposition through partial functional dependencies. *Data & Knowledge Engineering* 43(2), 2002, 207-234.

[6] P. De Bra and J. Paredaens. Conditional dependencies for horizontal decompositions. *ICALP*, 1983.

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

[8] M. Gyssens, L. Lakshmanan, and S. N. Subramanian. Tables as a paradigm for querying and restructuring. *PODS*, 1996.

[9] Qi He, Tok Wang Ling. *Extending and inferring functional dependencies in schema transformation: extended version.* Technical report, TRA3/04. School of Computing, National University of Singapore, 2004. http://www-appn.comp.nus.edu.sg/~esubmit/diglib/public/techrep/2004/TRA304/report.pdf

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

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

[12] L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. On efficiently implementing schemaSQL on SQL database system. *VLDB*, 1999, 471-482.

[13] L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. SchemaSQL—an extension to SQL for multidatabase interoperability. *TODS*, 2001, 476-519.

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

[15] Mong Li Lee and Tok Wang Ling. Resolving constraint conflicts in the integration of ER schemas. *ER*, 1997, 394-407.

[16] Chen Li. Describing and Utilizing Constraints to Answer Queries in Data-Integration Systems. *IIWeb*, 2003, 163-168.

[17] Tok Wang Ling. Extending classical functional dependencies for physical database design (lecture notes), 2001. http://www.comp.nus.edu.sg/~lingtw/cs4221/extended.fds.pdf

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

[19] R. J. Miller. Using schematically heterogeneous structures. *SIGMOD*, 1998, 189-200.

[20] R. Ramakrishnan and J. Gehrke. *Database Management Systems, 2nd Ed*. McGraw-Hill, 1999, 427, 438-444.

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

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

[23] M. W. W. Vermeer and P. M. G. Apers. The role of integrity constraints in database interoperation. *VLDB*, 1996, 425-435.

APPENDIX

A. A Complete Set of Inference Rules to Infer Qualified FDs in Fixed Schemas

Given a set of relation schemas $\mathcal{S}$ with the same set of attributes U in database DB, and a set of qualified FDs F holding in $\mathcal{S}$, let $\overline{X}$ be a mixed set of regular and restriction attributes from U ($\overline{X}$ may comprise only regular or restriction attributes), $Y \subseteq U$ and $Z \subseteq U$ be two sets of regular attributes, and $A \in U$; let $\mathcal{R}1 \subseteq \mathcal{R} \subseteq \mathcal{S}$, $S1 \subseteq S \subseteq dom(A)$, we have the following inference rules:

(1) If $\mathcal{R}(\overline{X} \rightarrow Y)$ holds, then $\mathcal{R}1(\overline{X} \rightarrow Y)$ holds.

(2) If $\{R_i, R_j\}(\overline{X} \rightarrow Y)$ holds for any $R_i, R_j \in \mathcal{R}$, then $\mathcal{R}(\overline{X} \rightarrow Y)$ holds.

(3) If $\mathcal{R}(\overline{X}, A_{\sigma=S} \rightarrow Y)$ holds, then $\mathcal{R}(\overline{X}, A_{\sigma=S1} \rightarrow Y)$ holds.

(4) If $\mathcal{R}(\overline{X}, A_{\sigma=\{a_i, a_j\}} \rightarrow Y)$ holds for any $a_i, a_j \in S$, then $\mathcal{R}(\overline{X}, A_{\sigma=S} \rightarrow Y)$ holds.

(5) If $a_i \in dom(A)$, then $\mathcal{R}(A_{\sigma=\{a_i\}} \rightarrow A)$ holds.

(6) If $\mathcal{R}(A_{\sigma=\{a_i\}}, \overline{X} \rightarrow Y)$ holds for each $a_i \in S$, then $\mathcal{R}(A_{\sigma=S}, A, \overline{X} \rightarrow Y)$ holds.

(7) If $Y \subseteq \overline{X}$, then $\mathcal{R}(\overline{X} \rightarrow Y)$.

(8) If $\mathcal{R}(\overline{X} \rightarrow Y)$ holds, then $\mathcal{R}(\overline{X}, Z \rightarrow Y, Z)$.

(9) If $\mathcal{R}(\overline{X} \rightarrow Y)$ and $\mathcal{R}(\overline{X1}, Y \rightarrow Z)$ hold, where $\overline{X1}$ is a set (possibly an empty set) of some restriction attributes in $\overline{X}$, then $\mathcal{R}(\overline{X} \rightarrow Z)$ holds.

(10) If $\mathcal{R}(\overline{X} \rightarrow Y)$ holds and A does not occur in the restriction attributes of $\overline{X}$, then $\mathcal{R}(\overline{X}, A_{\sigma=dom(A)} \rightarrow Y)$ holds.

(11) If $\mathcal{R}(\overline{X}, A_{\sigma=dom(A)} \rightarrow Y)$ holds, then $\mathcal{R}(\overline{X} \rightarrow Y)$ holds. □

B. Derived Rules to Infer Qualified FDs in Schema Transformation.

The rules in each algorithm below are used to infer qualified FDs in schematic discrepant transformation. These rules are derived from the inference rules in Appendix A and the propagation rules P1~P4 in Section 3. These derived rules apply to the qualified FDs of original relations, and produce the implied qualified FDs of the transformed relations, avoiding the computation of qualified FD closures. The rules are complete to compute reasonable qualified FDs in schematic discrepant transformation.

Algorithm INFER_SPLIT: *Inference of qualified FDs for a split operation.*

INPUT: Let $R(A_1, ..., A_n, B)$ be an original relation, and $b_i(A_1, ..., A_n)$, $i = 1, ..., m$, be the transformed relations using *split(R, B)*. Let F be a set (not necessary a closure) of qualified FDs on R.

OUTPUT: a set of qualified FDs, G, holding in the set of the transformed relations $\{b_1, ..., b_m\}$.

METHOD: Let $\overline{X}$ and $\overline{Y}$ be 2 mixed sets of regular and qualification attributes from $\{A_1, ..., A_n\}$, and $A \in \{A_1, ..., A_n\}$. We compute the qualified FDs in G using the following rules:

(1) If $R(\overline{X} \rightarrow A) \in F$, then $dom(B)(\overline{X} \rightarrow A) \in G$.

(2) If $R(\overline{X}, B \rightarrow A) \in F$, then $b_i(\overline{X} \rightarrow A) \in G$ for each $b_i \in dom(B)$.

(3) If $R(\overline{X} \rightarrow B) \in F$ and $R(\overline{Y}, B \rightarrow A) \in F$, then $dom(B)(\overline{X}, \overline{Y} \rightarrow A) \in G$. □

Algorithm INFER_UNITE: *Inference of qualified FDs for a unite operation.*

INPUT: Let $b_i(A_1, ..., A_n)$, $i = 1, ..., m$, be original relations, and $R(A_1, ..., A_n, B)$ be the transformed relations using $unite(\{b_1, ..., b_m\}, B)$. Let F be a set (not necessary a closure) of qualified FDs holding in the original relations $\{b_1, ..., b_m\}$.

OUTPUT: a set of qualified FDs, G, holding in the transformed relation R.

METHOD: Let $\overline{X}$ be a mixed set of regular and qualification attributes from $\{A_1, ..., A_n\}$, and $A \in \{A_1, ..., A_n\}$. We compute the qualified FDs in G using the following rules:

(1) If $dom(B)(\overline{X} \rightarrow A) \in F$, then $R(\overline{X} \rightarrow A) \in G$.

(2) If $\mathcal{R}_B(\overline{X} \rightarrow A) \in F$ for a set of relations $\mathcal{R}_B \subset \{b_1, ..., b_m\}$, then $R(B_{\sigma = \mathcal{R}_B}, \overline{X} \rightarrow A) \in G$. □

Algorithm INFER_UNFOLD: *Inference of qualified FDs for a set of unfold operations.*

INPUT: Let $R_i(A_1, ..., A_n, B, C)$ $(i=1,...,l)$ be original relations, and $S_i(A_1, ..., A_n, b_1, ..., b_m)$ be the transformed relations using $unfold(R_i, B, C)$ for each $i=1, ..., l$. Let F be a set (not necessary a closure) of qualified FDs holding in the set of the original relations $\{R_1, ..., R_l\}$.

OUTPUT: a set of qualified FDs, G, holding in the transformed relations $\{S_1, ..., S_l\}$.

METHOD: Let $\overline{X}$, $\overline{Y}$ and $\overline{Z}$ be mixed sets of regular and qualification attributes from $\{A_1, ..., A_n\}$, and $A \in \{A_1, ..., A_n\}$.

Let $\mathcal{R}$ and $\mathcal{S}$ be subsets of relations of $\{R_1, ..., R_l\}$ and $\{S_1, ..., S_l\}$ respectively; the relations of $\mathcal{S}$ are transformed from the relations of $\mathcal{R}$. We compute the qualified FDs in G using the following rules:

(1) If $\mathcal{R}(B_{\sigma=\{b_i\}}, \overline{X} \rightarrow C) \in F$, then $\mathcal{S}(\overline{X} \rightarrow b_i) \in G$.

(2) If $\mathcal{R}(B, \overline{X} \rightarrow C) \in F$ or $\mathcal{R}(\overline{X} \rightarrow C) \in F$, then $\mathcal{S}(\overline{X} \rightarrow b_i) \in G$ for each $b_i \in dom(B)$.

(3) If $\mathcal{R}(B_{\sigma=\{b_i\}}, \overline{X}, C \rightarrow A) \in F$, then $\mathcal{S}(\overline{X}, b_i \rightarrow A) \in G$.

(4) If $\mathcal{R}(B, \overline{X}, C \rightarrow A) \in F$ or $\mathcal{R}(\overline{X}, C \rightarrow A) \in F$, then $\mathcal{S}(\overline{X}, b_i \rightarrow A) \in G$ for each $b_i \in dom(B)$.

(5) If $\mathcal{R}(\overline{X} \rightarrow A) \in F$, then $\mathcal{S}(\overline{X} \rightarrow A) \in G$.

(6) If $\mathcal{R}(\overline{X} \rightarrow B) \in F$ and $\mathcal{R}(\overline{Y}, B \rightarrow A) \in F$, then $\mathcal{S}(\overline{X}, \overline{Y} \rightarrow A) \in G$.

(7) If $\mathcal{R}(\overline{X} \rightarrow C) \in F$ and $\mathcal{R}(\overline{Y}, C \rightarrow A) \in F$, then $\mathcal{S}(\overline{X}, \overline{Y} \rightarrow A) \in G$.

(8) If $\mathcal{R}(\overline{X} \rightarrow B) \in F$, $\mathcal{R}(\overline{Y}, B \rightarrow C) \in F$ and $\mathcal{R}(\overline{Z}, C \rightarrow A) \in F$, then $\mathcal{S}(\overline{X}, \overline{Y}, \overline{Z} \rightarrow A) \in G$.

(9) If $\mathcal{R}(\overline{X} \rightarrow C) \in F$, $\mathcal{R}(\overline{Y}, C \rightarrow B) \in F$ and $\mathcal{R}(\overline{Z}, B \rightarrow A) \in F$, then $\mathcal{S}(\overline{X}, \overline{Y}, \overline{Z} \rightarrow A) \in G$. □

Algorithm INFER_FOLD: *Inference of qualified FDs for a set of fold operations.*

INPUT: Let $R_i(A_1, ..., A_n, b_1, ..., b_m)$ $(i=1,...,l)$ be original relations, and $S_i(A_1, ..., A_n, B, C)$ be the transformed relations using $fold(R_i, B, C)$ for each $i=1, ..., l$. Let F be the set of (not necessary a closure) qualified FDs holding in the set of the original relations $\{R_1, ..., R_l\}$.

OUTPUT: a set of qualified FDs, G, holding in the transformed relations $\{S_1, ..., S_l\}$.

METHOD: Let $\overline{X}$ be a mixed set of regular and qualification attributes from $\{A_1, ..., A_n\}$, and $A \in \{A_1, ..., A_n\}$. Let $\mathcal{R}$ and $\mathcal{S}$ be subsets of relations of $\{R_1, ..., R_l\}$ and $\{S_1, ..., S_l\}$ respectively; the relations of $\mathcal{S}$ are transformed from the relations of $\mathcal{R}$. We compute the qualified FDs in G according to the following rules:

(1) If $\mathcal{R}(\overline{X} \rightarrow b_i) \in F$ then $\mathcal{S}(B_{\sigma=\{b_i\}}, \overline{X} \rightarrow C) \in G$.

(2) If $\mathcal{R}(\overline{X}, b_i \rightarrow A) \in F$ then $\mathcal{S}(B_{\sigma=\{b_i\}}, \overline{X}, C \rightarrow A) \in G$.

(3) If $\mathcal{R}(\overline{X} \rightarrow A) \in F$ then $\mathcal{S}(\overline{X} \rightarrow A) \in G$. □