THE NATIONAL UNIVERSITY
of SINGAPORE

School *of* Computing
Lower Kent Ridge Road, Singapore 119260

TRA3/04

Extending and Inferring Functional Dependencies in Schema Transformations:Extended Version

Qi HE and Tok Wang LING

March 2004

Technical Report

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

JAFFAR, Joxan
Dean of School

Extending and Inferring Functional Dependencies in Schema Transformation

Abstract

Functional dependency (FD) plays an important role in individual databases. In this paper, we study FDs in the context of multi-database interoperability. A major challenge in the integration of heterogeneous database schemas is of schematic discrepancies, when the data (values) of one database correspond to metadata (schema labels) of another. We first study the schematic discrepant transformations, i.e., transformations between schematic discrepant schemas. We then define "restricted FD", an extension to conventional FD, to formalize some class of constraints in schematic discrepant databases, and give a complete set of inference rules of restricted FDs. Then we study the propagation of restricted FDs during schematic discrepant transformations. Algorithms are proposed to derive all the restricted FDs in transformed schemas from restricted FDs in original schemas. At last, we show some applications of restricted FDs in the context of multi-database interoperability: (1) use FDs to verify whether a SchemaSQL view is well-defined, (2) use FDs to normalize transformed (integrated) schemas, and so on.

1 Introduction

Schema integration [Alb00, BLN86, LL95, LL96] 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 [SG90] (also called *multidatabase system* or *federated database system*), to integrate data in a data warehouse, or to integrate user views in database design. A major challenge in schema integration is *schematic discrepancy*. 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 [KLK91, Mil98].

Example 1.1: In Figure 1.1, the 4 databases use different schemas to record the same information: *supplying prices of products in different months.* In *DB1*, all information (products, suppliers, months and prices) is modeled as attribute values; in *DB2*, the months of *Jan, ..., Dec* become attribute names whose values are prices; in *DB3*, the months become relation names; in *DB4*, the months are modeled as attribute names, and suppliers as relation names.□

DB1:

Supply

product	supplier	month	price
P1	s1	Jan	105
P1	s1	Dec	110
P1	sn	Jan	97
P1	sn	Dec	99
…	…	…	…

DB2:

Supply

product	supplier	Jan	…	Dec
P1	s1	105	…	110
P1	sn	97	…	99
…	…	…	…	…

DB3:

Jan

product	supplier	price

…

Dec

product	supplier	price

p1	s1	105
p1	sn	97
…	…	…

p1	s1	110
p1	sn	99
…	…	…

DB4:

s1

product	Jan	…	Dec
p1	105	…	110
…	…	…	…

…

sn

product	Jan	…	Dec
p1	97	…	99
…	…	…	…

Figure 1.1: Schematic discrepancy: months and suppliers modeled differently in *DB1~DB4*

In data warehouse world, people separate *dimensional attributes* (e.g., *product*, *month* and *supplier* in *DB1*) from *measure attributes* (e.g., *price* in *DB1*). We adapt the terms here and assume dimensional attributes' values would be modeled as schema labels or attribute values in discrepant schemas, while measure attributes' values could only be attribute values.

To integrate or interoperate with schematic discrepant databases, people need to transform discrepant schemas into a unified form. We call a transformation between schematic discrepant schemas a ***schematic discrepant transformation*** (or just *transformation*). In Figure 1.1, the process of transforming any database into another is such a transformation.

Recently, People have studied how to transform and query discrepant data [KLK91, LSS01, LSS99] and how to use schematic discrepancy [ASX01, Mil98]. In this paper, we focus on the inference of integrity constraints (ICs), specifically a super class of FDs, in transformations, and the application of those ICs to works in schematic discrepancy. Some previous works have studied the derivation of ICs for integrated schema [LL97, RPG95], the roles of ICs in database interoperation [VA96], and semantic query optimization for query plans of multidatabase systems [HK00] in general. But they have failed to consider schematic discrepancy in schema integration, neither have they proven the completeness of their methods.

1.1 Extending FDs in Schematic Discrepancy

To formalize and reason about FD-like ICs in transformations, we propose *restricted FDs*, an extension to conventional FDs which are not powerful enough to express meaningful ICs in schematic discrepant databases. A formal definition of restricted FD will be given in Section 3.1. The following example shows some intuitions of our proposal.

Example 1.2: Suppose we have an IC in *DB1~DB4* (Figure 1.1): (**C1**) "*in each month, a product supplied by supplier s1 or s2 has the same price.*" We'll express C1 in different databases as follows.

DB1: $Supply(product, month, supplier_{\sigma=\{s1, s2\}} \rightarrow price)$

DB2: $Supply(product, supplier_{\sigma=\{s1, s2\}} \rightarrow Jan, \ldots, Dec)$

DB3: $mi(product, supplier_{\sigma=\{s1, s2\}} \textcircled{R} price)$, for each $mi \in \{Jan, \ldots, Dec\}$

DB4: $\{s1, s2\}(product \rightarrow Jan, \ldots, Dec)$

These dependencies can be interpreted as follows:

DB1: in the sub-relation $\sigma_{supplier \in \{s1, s2\}} Supply$, the FD *product, month→price* holds.

DB2: in the sub-relation $\sigma_{supplier \in \{s1, s2\}} Supply$, the FD *product→Jan, …, Dec* holds.

DB3: for each relation name $mi \in \{Jan, \ldots, Dec\}$, in the sub-relation $\sigma_{supplier \in \{s1, s2\}} mi$, the FD *product→price* holds.

DB4: in the union of the two relations $s1 \cup s2$, the FD *product→Jan, …, Dec* holds.□

There're two differences between a restricted FD and a conventional FD: (1) a restricted FD may hold over a set of relations (e.g., the dependency in *DB4* in Example 1.2) or individual relations. And therefore we should explicitly express the relation set or relation on which a restricted FD holds; (2) a restricted FD may hold in a sub-relation (e.g., the dependencies in *DB1~DB3*).

1.2 Paper Overview

Inferring and using restricted FDs in schematic discrepant transformations is the main objective of this paper. In Section 2, we introduce how to implement a transformation using four restructuring operators, based on the works of [GLS96, LSS99]. The main contributions of this paper are in Section 3 ~ 6:

In Section 3, we study schematic discrepant transformation itself, i.e., lossless and non-redundant transformations. Intuitively, a lossless transformation preserves all information during the transformation. A non-redundant transformation uses

minimum number of transformation steps. In Section 4, we give the definition and inference rules of *restricted FD*. We also prove the completeness of the inference rules. In Section 5, we study the propagation of restricted FDs in transformations. To this end, we first give the rules on how restricted FDs change in application of each kind of restructuring operators in Section 5.1. Then in Section 5.2, we propose algorithms to compute restricted FDs in a transformation consisting of a sequence of restructuring operators. In Section 5.3, we prove that our algorithms are sound and complete in the sense that they compute all the restricted FDs in transformed relations from restricted FDs in original relations. We study the time complexity of our algorithms in Section 5.4. Though the problem of inferring restricted FDs in schematic discrepant transformations is inherently exponential, our algorithm behaves polynomially for several classes of inputs. In Section 6, we show some applications of restricted FDs in the context of multi-database interoperability. We conclude the paper in Section 7.

Notations: The notation *DB::R* represents a relation *R* in a database *DB*, following the SchemaSQL convention. Capital letters near the beginning of the alphabet 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. *dom(A)* stands for the domain of an attribute *A*.

2 Schematic Discrepant Transformation

In this section, we study schematic discrepant transformation itself. Section 2.1 gives the definitions of restructuring operators. Section 2.2 introduces how to implement a transformation using those operators. Section 2.3 is a survey of SchemaSQL language for multi-database interoperability.

2.1 Restructuring Operators

We now give the formal definitions of 4 kinds of restructuring operators adapted from [LSS99], originally introduced in the context of the *tabular algebra* [GLS96]. They will be used to implement schematic discrepant transformations. Figure 2.1 shows the applications of those operators in the transformations between the databases in Figure 1.1.

As is the case in real-life tables, our relations need not have entries for every row and column combination. To deal with possible absence of entries, we introduce the *no-value* "-" denoting some attribute is inapplicable for a tuple.[1]

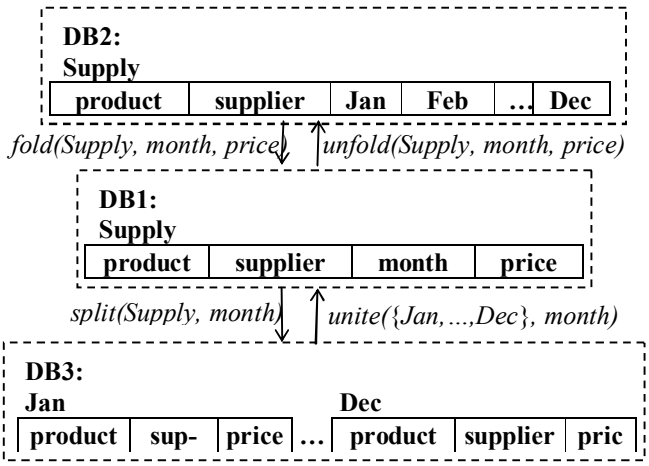


Figure 2.1: Restructuring operators

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

$S = \{(a_1, ..., a_n, c_1, ..., c_m) \mid (c_i \neq \text{"-"} \ \& \ (a_1, ..., a_n, b_i, c_i) \in R)$ or $(c_i = \text{"-"} \ \& \ (\neg\exists$ a tuple $t \in R$: $t[A_1, ..., A_n, B] = (a_1, ..., a_n, b_i))), 1 \leq i \leq m\}$.

[1] In relational databases, people use *null* to represent *no-value* or *unknown-value*. However, in schematic discrepant transformation, it is better to distinguish these two special values.

Example 2.1: In Figure 2.2, let *R2 = unfold(R1, supplier, price)*. In *R2*, "-" is used as not exist a tuple $t \in R1$, such that t[*product, supplier*] = (*p2, s2*). That is, supplier *s2* does not supply *p2*. □

R1

product	supplier	price
p1	s1	100
p1	s2	105
p2	s1	200

R2

product	s1	s2
p1	100	105
p2	200	-

Figure 2.2: *unfold* operator generates *no-value* ("-") in *R2*

***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 of attribute B, and all entries in columns $b_1, ..., b_m$ of R are from $dom(C) \cup \{-\}$, such that B is a dimensional attribute and C is a measure attribute, and $B, C \notin \{A_1, ..., A_n\}$. *fold(R, B, C)* transforms R into a relation $S(A_1, ..., A_n, B, C)$, defined as:
$S = \{(a_1, ..., a_n, b_i, c_i) \mid \exists\, t \in R: t[A_1, ..., A_n] = (a_1, ..., a_n) \,\&\, t[b_i] = c_i \,\&\, c_i \neq \text{"-"}\}$.

To simplify the presentation, in this paper, we assume only attributes b_i's which are values of some dimensional attribute B could take *no-value*.

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

$$b_i = \{t[A_1, ..., A_n] \mid t \in R \,\&\, t[B] = b_i\}.$$

***unite(R_B, B)*:** Let $R_B = \{b_1, ..., b_m\}$ be a set of relations in a given database, such that each relation name b_i ($i=1,...,m$) is a value of a dimensional attribute B, and all the relations have a common schema $b_i(A_1, ..., A_n)$. *unite(R_B, B)* transforms R_B into a relation $S(B, A_1, ..., A_n)$, defined as:

$$S = \{t \mid \exists\, t' \in b_i : t[A_1, ..., A_n] = t'[A1, ..., An] \,\&\, t[B] = b_i\}.$$

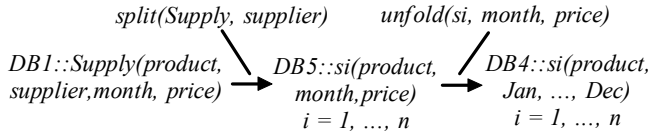
2.2 Implementation of Transformations Using Restructuring Operators

In general, a schematic discrepant transformation can be implemented using a sequence of restructuring operators:

$$R_0 \xrightarrow{T_1} R_1 \xrightarrow{T_2} \ ... \ \xrightarrow{T_k} R_k$$

Each R_i, $i=1, ..., k$, represents a set (possibly a singleton set) of relations. R_0 and R_k are the sets of original and target transformed relations; $R_1, ..., R_{k-1}$ are the sets of intermediate transformed relations. Each T_i represents one (or a set of) restructuring operator(s) which transform the relations of R_{i-1} into the relations of R_i.

Example 2.2: In Figure 1.1, we can transform the relation of *DB1* into the relations of *DB4* as follows: (*DB5* is an intermediate result not given in Figure 1.1)



That is, first transform the relation *DB1::Supply* into the set of relations {*s1, ..., sn*} of *DB5* using a *split* operator; then transform the relations of *DB5* into *DB4* by applying *unfold* on each relation *si* of *DB5*. Note n *unfold* operators are used in the 2nd step. □

A transformation could be implemented by more than one sequence of operators. We call two transformations T and T' are ***equivalent*** if they are performed on the same set of relation schemas R, and for any instance r of R, $T(r)=T'(r)$.

For example, in Example 2.2, we can transform *DB1* into *DB4* using another way: first transform *DB1* into *DB2* (Figure 1.1) with *unfold(Supply, month, price)*, then transform *DB2* into *DB4* with *split(Supply, supplier)*.

Generally, given a set of original relation R_0 and a set of target transformed relations R_k which are schematic discrepant from R_0, we can implement the transformation using restructuring operators in 2 phases:

(1) First perform *fold* and/or *unite* operators to transform R_0 into an intermediate relation, say R_j, such that schema labels in R_0 (which are modeled as attribute values or different kinds of schema labels in R_k) become attribute values in R_j.

(2) Then perform *unfold* and/or *split* operators to transform R_j into R_k, such that certain attributes' values in R_j become schema labels in R_k.

Note not all transformations need both the 2 phases. Example 2.2 only has Phase 2.

In this paper, we assume a transformation could include several *fold/unfold* operators (or operator sets) but one *unite/split* operator at most. In practice, a transformation would have multiple *fold/unfold* operators (or operator sets) if relation schemas contain multiple *measure attributes*. Furthermore, the results of this paper can be extended readily to more complex transformations, e.g.,

- Transformations include multiple *unit/split* also. Then a relation name should model values of multiple attributes being united (or split).
- Transformations include not only the restructuring operators, but also the classical relational algebra. For example, in [LSS99], Lakshmanan et al proposed a logical algebra for SchemaSQL (see next sub-section) consisting of the relational algebra and the four restructuring operators. We'll discuss this extension in Section 6.1.

2.3 SchemaSQL Language

In this subsection, we briefly overview Lakshmanan et al's work, SchemaSQL language [LSS01], which has been used to solve a broad range of problems recently [LSS99, Mil98]. SchemaSQL is an extension to SQL for enabling multi-database interoperability. It treats data and schema labels in a uniform manner, i.e., variables can range data and schema labels, which facilitates the interoperability among schematic discrepant databases. In what follows, we explain the syntax and semantics of SchemaSQL through an example. And in Section 6.1, we'll try to solve a problem of SchemaSQL by using FDs.

Example 2.3: In Figure 1.1, we can transform *DB3* into *DB2* using the following view definition *Q*:

create view DB2::Supply(product, supplier, M)
select T.product, T.supplier, T.price
from DB3 → M, DB3::M T

There're 3 types of variables in *Q*, i.e., relation-name variable *M* ranging over relation names in *DB3*, tuple variable *T* ranging over tuples of relation *DB3::M* for each instantiation of *M,* and domain variables *T.product, T.supplier* and *T.price*. The semantics of this view definition is interpreted in 4 steps:

(1) Compute the valid instantiations of the variables in *Q* (Figure 2.3). The values of variable *T* represent tuple ids in a relation.

(2) Determine the schema of the view. Since there is a variable *M* in the "*create view*" clause of *Q*, it creates a dynamic view whose schema depends on the instantiations of *M*. In this example, *M* can be *Jan, ..., Dec*, so the view schema is:
DB2::Supply(product, supplier, Jan, ..., Dec).

(3) Allocate the instantiations of Figure 2.3 according to the view schema (Figure 2.4). Each tuple in Figure 2.3 becomes one tuple in the allocated table. *no-value* ("-") is used to fill each place without any value.

(4) Merge the tuples in the allocated table, and get the target relation, *DB2::Supply* in Figure 1.1. According to [LSS01], 2 tuples are merge-able if for each attribute, either the 2 values in the 2 tuples are the same, or at least one value is "-". E.g., in Figure 2.4, the 1st tuple can be merged with the 3rd tuple. However this step is problematic, which may cause ambiguous results. We'll study this problem in Section 6.1. □

M	T	T.product	T.supplier	T.price
Jan	t1	p1	s1	105
Jan	t2	p1	sn	97
Dec	t1	p1	s1	110
Dec	t2	p1	sn	99
…	…	…	…	…

Figure 2.3: valid instantiations of variables in *Q*

product	supplier	Jan	…	Dec
p1	s1	105	…	-
p1	sn	97	…	-
p1	s1	-	…	110
p1	sn	-	…	99
…	…	…	…	…

Figure 2.4: allocated table of *Q*

3 Lossless and Non-redundant Transformations

In Section 3.1, we give some algebraic laws for restructuring operators. In Section 3.2, we study the lossless and non-redundant transformations. We also give an algorithm to simplify a transformation into a non-redundant one using algebraic laws of restructuring operators.

3.1 Algebraic Laws of Restructuring Operators

Like many other algebra, there are laws that the restructuring operators obey. In the following, we give two kinds of laws, reconstructibility and commutativity, of restructuring operators. These laws can be used to simplify a transformation implemented using a sequence of restructuring operators.

We first see the reconstructibility. That is, relations being transformed with an *unite* (*fold*) operator can be recovered with a *split* (*unfold*) operator, and vice versa.

Theorem 3.1 (*Reconstructibility of split/unite*): Given a relation $R(A_1, ..., A_n, B)$, let $b_i(A_1, ..., A_n)$ ($i = 1, ..., m$) be the transformed relations using *split(R, B)*, i.e., the distinct values of B in R, $\{b_1, ..., b_m\}$, become relation names of the transformed relations. Then:

$$unite(split(R, B), B)=R$$
$$split(unite(\{b_1, ..., b_m\}, B), B) = \{b_1, ..., b_m\} \quad \square$$

Theorem 3.2 (*Reconstructibility of unfold/fold*): Given a relation $R(A_1, ..., A_n, B, C)$, let $S(A_1, ..., A_n, b_1, ..., b_m)$ be the transformed relation using *unfold(R, B, C)*, i.e., the values of B in R, $b_1, ..., b_m$, become attribute names in S, and the values of C in R become the values of attributes $b_1, ..., b_m$ in S. Then:

If the FD $A_1, ..., A_n, B \to C$ holds in R, then $fold(unfold(R, B, C), B, C)= R$
If the FD $A_1, ..., A_n \to b_1, ..., b_m$ holds in S, then $unfold(fold(S, B, C), B, C)= S$ □

We omit the formal proofs of the above 2 theorems, but explain the reconstructibility of *unfold* and *fold* a little. In general, an operation is reconstructible if it defines a one to one mapping from original relations onto transformed relations. For *unfold*, in the absence of the FD $A_1, ..., A_n, B \to C$ holding in R, the operation may produce one of several results. That is, an *unfold* operation is a one to many mapping without that FD. On the contrary, a *fold* operation is a many to one mapping from original relations onto transformed relations without the FD $A_1,...,A_n \to b_1,...,b_m$ holding in S. The following example explains this.

Example 3.1 (Figure 3.1): Suppose in the relation schema R, the attribute names *b1, b2* are values of a fixed attribute B, and the values of *b1,b2* are values of a fixed attribute C. Suppose the FD $A \to b1, b2$ does not hold in R.

We can transform either of the two instances I1, I2 of R into the same relation of S by applying *fold(R, B, C)*. That is, the mapping from the original relations onto the transformed relations is many to one, which makes the recovering impossible. □

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 3.1: Un-recoverable fold operator

We have observed that for practical case an *unfold* or *fold* operation makes sense only if the FDs mentioned above holds. Henceforth, we make this a requirement for the *unfold* and *fold* operations.

Next, let's see the other kind of property, commutativity, of restructuring operators. That is, operators can commute in a transformation if they are performed on different attributes.

Theorem 3.3 (*Commutativity of unfold operators*): Given a relation $R(A_1, ..., A_n, B1, C1, B2, C2)$, if the FDs $A_1, ..., A_n, B1, C1, B2 \to C2$ and $A_1, ..., A_n, B1, B2, C2 \to C1$ hold, then we have:

$$unfold(unfold(R, B1, C1), B2, C2) = unfold(unfold(R, B2, C2), B1, C1). \square$$

Theorem 3.4 (*Commutativity of fold operators*): Given a relation $R(A_1, ..., A_n, b_{11}, ..., b_{1m}, b_{21}, ..., b_{2l})$, where $b_{11}, ..., b_{1m}$ are values from $dom(B1)$, and all values appearing the columns $b_{11}, ..., b_{1m}$ are from $dom(C1) \cup \{-\}$, and $b_{21}, ..., b_{2l}$ are values from $dom(B2)$, and all values appearing the columns $b_{21}, ..., b_{2l}$ are from $dom(C2) \cup \{-\}$, if the FDs $A_1, ..., A_n, b_{21}, ..., b_{2l} \to b_{11}, ..., b_{1m}$ and $A_1, ..., A_n, b_{11}, ..., b_{1m} \to b_{21}, ..., b_{2l}$ hold, then we have:

$$fold(fold(R, B1, C1), B2, C2) = fold(fold(R, B2, C2), B1, C1). \square$$

Theorem 3.5 (*Commutativity of unfold and split*): Given a relation $R(A_1, ..., A_n, B1, B2, C2)$, if the FD $A_1, ..., A_n, B1, B2 \to C2$ holds, then we have:

$$split(unfold(R, B2, C2), B1) = \{unfold(b_{1i}, B2, C2)) \mid b_{1i} \in split(R, B1)\} \square$$

Theorem 3.6 (*Commutativity* of *fold and unite*): Given a set of relations $\{b_{1i}(A_1, ..., A_n, b_{21}, ..., b_{2l}) \mid i=1, ..., m\}$ such that each relation name $b_{1i} \in dom(B1)$ for some attribute $B1$, and $b_{21}, ..., b_{2l}$ are values from $dom(B2)$ and all values appearing the columns $b_{21}, ..., b_{2l}$ are from $dom(C2) \cup \{-\}$ for some attributes $B2, C2$, if the FD $A_1, ..., A_n \to b_{21}, ..., b_{2l}$ holds in each relation of b_{1i}, then we have:

$$unite(\{fold(b_{1i}, B2, C2) \mid i=1,...,m\}, B1) = fold(unite(\{b_{11}, ..., b_{1m}\}, B1), B2, C2). \square$$

Example 3.2: Suppose in Figure 1.1, we want to transform the set of relation $\{s1, ..., sn\}$ of *DB4* into the *Supply* relation of *DB1*. Suppose in *DB4*, the FD $product \to Jan, ..., Dec$ holds in each relation si. Then the following two transformations are equivalent to each other according to Theorem 3.6:

$Supply = unite(\{fold(si, month, price) \mid i=1,...,n\}, supplier)$, or

$Supply = fold(unite(\{s1, ..., sn\}, supplier), month, price). \square$

3.2 Lossless and Non-redundant Transformations

In practice, one is mostly interested in semantics-preserving transformations, i.e. transformations such that both original and transformed relations represent exactly the same real world facts, though with a different syntax. A relation (or relation set) can be losslessly converted into another relation (or relation set), and conversely, hence the name of *lossless transformation.*

Definition 3.1 (*Lossless transformation*): Given a schematic discrepant transformation T, let R be the schema of the original relation (or relation set) of T. If there exist an inverse transformation T' of T, such that for any instance r of R,

$$T'(T(r))=r$$

then T is called a *lossless transformation*. □

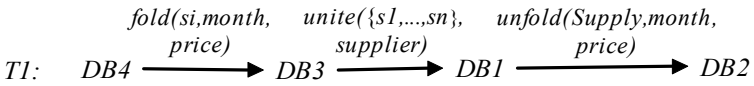
Theorem 3.7: Given a schematic discrepant transformation T consisting of a sequence of restructuring operators, if each of the *unfold* and *fold* operators in T satisfies the reconstructibility, then T is a lossless transformation. □

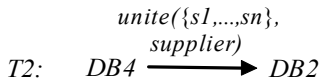
A lossless transformation may be redundant because unnecessary operators are used. And using reconstructibility and commutativity of restructuring operators, we can simplify a redundant transformation. In what follows, we first define non-redundant transformations (i.e. transformations using minimum number of transformation steps), then give an algorithm to simplify a transformation into a non-redundant one.

Definition 3.2 (*Non-redundant transformation*): Given two transformations T and T', we define a partial order "$\preceq$" as follows: $T' \preceq T$ provided $\forall t' \in T' \; \exists \, t \in T$: t' and t are of the same kind of operators, and have the same attribute names as parameters. A transformation T is non-redundant provided

$$\forall T' \preceq T \;\&\; T' \text{ is equivalent to } T \Rightarrow T \preceq T'. \square$$

Example 3.3: Suppose in Figure 1.1, the FD $product, supplier \to Jan, Feb, ..., Dec$ holds in *DB2::Supply*. We have two ways to transform *DB4* into *DB2*:





T1 is not a non-redundant transformation as we can find *T2* which is equivalent to *T1*, and $T2 \leq T1$ but not the converse. □

Algorithm SIMPLIFY_TRANSFORMATION

INPUT: A transformation $T = <T_1, ..., T_k>$, where each T_i represents one (or a set of) lossless restructuring operator(s)

OUTPUT: A non-redundant transformation *T'* equivalent to *T*

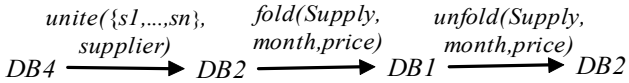
METHOD: We swap restructuring operators to bring together converse operators by the commutativity of restructuring operators, then remove redundant operators by the reconstructibility of restructuring operators.

Without loss of generality, we assume $T_1, ..., T_j$ are *fold* and *unite* operators, and $T_{j+1}, ..., T_k$ are *unfold* and *split* operators. (Recall in Section 2.2, we have mentioned a transformation can be divided into these two phases.)

1 **While** there exist two sets of operators of T_{i1} and T_{i2} which do the converse transformations but are performed on the same attributes **do**
2 **For** $i = i1$ to $j-1$
3 Commute T_i and T_{i+1} using commutativity of *fold* and *unite* operators (i.e. Theorem 3.4 or 3.6);
4 **For** $i = i2$ to $j+2$
5 Commute T_i and T_{i-1} using commutativity of *unfold* and *split* operators (i.e. Theorem 3.3 or 3.5);
6 Remove T_j and T_{j+1} from *T*. □

The following example explains the above algorithm using Example 3.3.

Example 3.4: Given the transformation *T1* of Example 3.3, according to Theorem 3.6, we can commute the set of *fold(si, month, price)* operators with *unite({s1,...,sn}, supplier)* and get the following transformation:



Then both the *fold* and *unfold* operators can be removed according to the reconstructibility of *fold*. Consequently only the *unite* operator is needed in this transformation. □

In the rest of this section, we'll prove the correctness of Alg SIMPLIFY_TRANSFORMATION.

Lemma 3.1: The transformation *T'* produced by Alg. SIMPLIFY_TRANSFORMATION is equivalent to *T*, and does not include any converse restructuring operators performed on the same attributes.

Proof: In each iteration of the while loop (Line 1~6), the algorithm removes T_{i1} and T_{i2} from *T*, and generates a transformation which is equivalent to *T*. This follows from the correctness of Theorem 3.3~3.6 readily. Note in Line 2~3, we can always move T_{i1} to T_j using commutativity of *fold* and *unite* operators, as the operators of T_i and T_{i+1} ($i = i1, ..., j-1$) are performed on different attributes. The reason is that we cannot perform *fold* on the same attributes twice or perform *fold* and *unite* on the same attribute during the phase from T_1 to T_j. Similarly, in Line 4~5, we can always move T_{i2} to T_{j+1}, and then remove T_j and T_{j+1} together. The while loop (Line 1) terminates when all the converse operator pairs in *T* are removed. □

Lemma 3.2: A lossless transformation *T* is a non-redundant transformation iff *T* does include any converse restructuring operators performed on the same attributes.

Proof: (=>) If *T* is a non-redundant transformation, then *T* does not include any pair of converse restructuring operators performed on the same attributes. Otherwise, we can always simplify *T* into an equivalent transformation by removing those converse operator pairs from *T* using Alg. SIMPLIFY_TRANSFORMATION.

(<=) Suppose *T* does not include any converse operators performed on the same attributes, but *T* is not a non-redundant transformation. That is $\exists\ T' \leq T$, *T'* is equivalent to *T*, but $T \leq T'$ is not true. That is, for some $T_i \in T$, $\neg\exists\ T'_j \in T'$, such that

T_i and T'_j are of the same kind of operators, and have the same attribute names as parameters. In this case, we can show that no matter what kind of operator T_i is, T' will not be equivalent to T. In what follows, we show this for a *unite* operator T_i:

Suppose $unite(R_B, B) \in T$, but $unite(S_B, B) \notin T'$ for 2 sets of relations R_B, S_B whose names are values of an attribute B. So the set of original relations of T have names of $b_1, \ldots, b_m$ which are values from $dom(B)$. As $unite(R_B, B) \in T$, $split(R, B) \notin T$ for some relation name R. Then in the target transformed relations of T, the values of B are modeled as attribute names or values, but not relation names. However in T', as we never perform *unite* operator on the B attribute, the values of B are modeled as relation names in the target transformed relations. The transformed relations of T and T' are different. So T' is not equivalent to T. This contradicts our assumption.

The proofs when T_i is one of the other 3 kinds of operators are similar, and therefore omitted by us. We conclude by contradiction that if T does not include any converse operators performed on the same attributes, then T is a non-redundant transformation. □

Intuitively, in a non-redundant transformation which is lossless, we never apply *unite* on some attribute B, and apply *split* on the same attribute later on, so do *fold* and *unfold*. From Lemma 3.1 and 3.2, we know that:

Theorem 3.8: Alg. SIMPLIFY_TRANSFORMATION correctly simplifies a lossless transformation into a non-redundant one. □

Lossless transformations are information preserving, which is always required in practice. And we can always simplify a lossless transformation into a non-redundant one. In the rest of the paper, we make the assumption that all the schematic discrepant transformations are lossless and non-redundant.

4 Restricted FD

In this section, we introduce *restricted FD*, an extension to conventional FD, to facilitate the expression and inference of FD-like ICs in schematic discrepant databases. We first give the formal definition of restricted FDs in Section 4.1. Then in Section 4.2, we'll propose inference rules of restricted FDs which are sound and complete. In Section 4.3, we'll give a method to compute an attribute closure w.r.t a set of restricted FDs. At last, in Section 4.4, we consider the impact of *no-value* on restricted FDs.

4.1 Definition of Restricted FD

In this sub-section, we define the syntax and semantics of *restricted FD.*

Definition 4.1 (*Restricted FD*): In general, given a database *DB*, and a set of relations S with the same set of attributes U in *DB*, we can represent a restricted FD as:

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

Syntax of the restricted FD:

(1) $R \subseteq S$ represents the relation(s) over which the restricted FD holds.

(2) $A_i \sigma{=}S_i$, $i=1, \ldots, n$, $A_i \in U$ and $S_i \subseteq dom(A_i)$, tells the ranges of attributes' values within which the restricted FD holds. For easy to reference, we call each $A_i \sigma{=}S_i$ a *restriction attribute* from U.

(3) $X \subseteq U$ and $Y \subseteq U$. For easy to reference, we call each attribute in X or Y a *regular attribute*.

Semantics of the restricted FD:

Given two tuples *t1*, *t2* from the relation(s) of R, we call the given restricted FD holds if the following holds:

If $t1.A_i \in S_i$ and $t2.A_i \in S_i$, for each $i = 1, \ldots, n$, 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 *restricted FD*. □

Readers could get intuitions of this concept from Example 1.2. To simplify the presentation, in this paper, we assume only dimensional attributes could be restriction attributes. In general, if a restricted FD

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

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

For convenience, in a restricted FD, "$\overline{X}, \overline{Y}$" is a shorthand for $\overline{X} \cup \overline{Y}$. Note for any attribute A, if $A_{\sigma=S1} \in \overline{X}$ and $A_{\sigma=S2} \in \overline{Y}$, then the two restriction attributes should be merged into $A_{\sigma=S1 \cap S2}$ in $\overline{X} \cup \overline{Y}$.

Generally speaking, ICs must be identified and specified based on the semantics of the real-world enterprise modeled. By looking at an instance of a relation (or a relation set), we might be able to tell that a certain restricted FD does not hold. However, we can never deduce that a restricted FD does hold by looking at one or more instances of the relation (or relation set) because a restricted FD, like other ICs, is a statement about all possible legal instances of the relation (or relation set).

4.2 Reasoning about Restricted FDs

In this sub-section, we first give a set of inference rules for restricted FDs, then prove the completeness of these rules.

4.2.1 Inference Rules of Restricted FDs

In this subsection, we study the reasoning about restricted FDs. In general, let F be a set of restricted FDs for a set of relation schemas R, and let f be a restricted FD. We say F *logically implies* f, if every instance of R that satisfies the dependencies in F also satisfies f. We define F^+, the ***closure of F***, to be the set of restricted FDs that are logically implied by F.

To understand logical implications among restricted FDs in general, we need to compute F^+ from F. We provide a complete set of inference rules, meaning that from a given set of restricted FDs F, the rules allow us to deduce all the true restricted FDs, i.e., those in F^+. Note in Section 5.1, we'll give another kind of inference rules (Axiom 5.1 and 5.2) to infer restricted FDs in application of restructuring operators.

Axiom 4.1: Given a set of relation schemas S with the same set of attributes U in database DB, and a set of restricted FDs F holding in S, let $\overline{X}$ be a mixed set of regular and restriction attributes from U ($\overline{X}$ may comprise only regular or restriction attributes) [2], $Y \subseteq U$ and $Z \subseteq U$ be two sets of regular attributes, and $A \in U$; let $R' \subseteq R \subseteq S$, $S' \subseteq S \subseteq dom(A)$, we have the following inference rules:

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

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

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

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

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

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

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

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

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

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

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

The proof of the soundness of these rules is easy and suppressed. Note rules (7) to (9) correspond to the reflexivity, augmentation and transitivity of Armstrong's Axioms [RG99], the inference rules of FDs, which are extended to infer restricted FDs here. In what follows, we give an example to apply Axiom 4.1.

Example 4.1: In Example 1.2, the dependency in *DB1* is equivalent to:
$Supply(month_{\sigma=dom(month)}, month, product, supplier_{\sigma=\{s1, s2\}} \rightarrow price)$ (rules (10), (11)), or
$Supply(month_{\sigma=\{mi\}}, month, product, supplier_{\sigma=\{s1, s2\}} \rightarrow price)$ for each $mi \in \{Jan, ..., Dec\}$ (rules (3), (6)), or
$Supply(month_{\sigma=\{mi\}}, product, supplier_{\sigma=\{s1, s2\}} \rightarrow price)$ (rules (5), (7) to (9)). □

We define the equivalences among sets of restricted FDs: Let F and G be 2 sets of restricted FDs. We say F and G are ***equivalent*** if $F^+ = G^+$. That is, each restricted FD in F is also in G^+, and each restricted FD in G is also in F^+.

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

Given a set of restricted FDs F, often we want to know whether a particular restricted FD $R\ (W, X \circledR Y)$ follows from F, where W is a set of restriction attributes, and X and Y are two sets of regular attributes. The solution is to compute $\boldsymbol{X^{+}_{(R, W)}}$, ***the closure of X under the restriction of W in*** R *with respect to* F, i.e., the set of attributes A such that $R\ (W, X \circledR A)$ can be deduced from F by Axiom 4.1. In Section 4.3, we'll give an algorithm to compute an attribute closure w.r.t a set of restricted FDs.

4.2.2 Completeness of the Inference Rules

In this sub-section, we'll try to prove the completeness of the rules in Axiom 4.1. We first define a "$\prec$" relation (called satisfaction relation) between two mixed sets of regular and restriction attributes $\overline{X}$ and $\overline{Y}$. Intuitively, $\overline{X} \prec \overline{Y}$ (called $\overline{X}$ satisfies $\overline{Y}$) if $\overline{X}$ defines more restricted conditions than $\overline{Y}$. The formal definition is as follows:

Definition 4.2: Given two mixed sets of regular and restriction attributes $\overline{X}$ and $\overline{Y}$, we define $\overline{X} \prec \overline{Y}$ provided: (a) for each restriction attribute $A_{\sigma=S1}$, S1 $\subset$ *dom(A)*, in $\overline{Y}$, there's a restriction attribute $A_{\sigma=S2}$ in $\overline{X}$ such that S2 $\subseteq$ S1, and (b) for each regular attribute A in $\overline{Y}$, either $A \in \overline{X}$ or $A_{\sigma=\{a\}} \in \overline{X}$ for some $a \in$ *dom(A)*. □

The following rules hold for the "$\prec$" relation:

(1) *Transitivity.* $\overline{X} \prec \overline{Y}$, $\overline{Y} \prec \overline{Z} \Rightarrow \overline{X} \prec \overline{Z}$

(2) *Reflexivity.* $\overline{X} \cup \overline{Y} \prec \overline{X}$

(3) *Augmentation.* $\overline{X} \prec \overline{Y}$, $\overline{X} \prec \overline{Z} \Rightarrow \overline{X} \prec \overline{Y} \cup \overline{Z}$

Lemma 4.1: If $R1 \subseteq R2$ and $W1 \prec W2$, then $X^{+}_{(R1, W1)} \supseteq X^{+}_{(R2, W2)}$.

Proof: For each attribute $A \in X^{+}_{(R2, W2)}$, $R2(W2, X \circledR A)$ holds. As $R1 \subseteq R2$ and $W1 \prec W2$, according to the rules (1, 3, 5, 7, 8, 9) of Axiom 4.1, we can derive $R1(W1, X \circledR A)$, and therefore $A \in X^{+}_{(R1, W1)}$. □

Given a mixed set of regular and restriction attributes $\overline{X}$ and a set of regular attributes Y, we define **the projection of $\overline{X}$ onto Y** as:

$$\overline{X}[Y] = \{A \mid A \in \overline{X} \cap Y\} \cup \{A_{\sigma=S} \mid A_{\sigma=S} \in \overline{X} \text{ and } A \in Y\}.$$

The inference rules of Axiom 4.1 are complete. We prove this by showing that if F is the given set of restricted FDs holding in S, and f is a restricted FD which cannot be proved by Axiom 4.1, then there must be an instance of S in which the dependencies of F all hold but f does not; that is, F does not logically imply f. The following algorithm is proposed to construct such an instance of S for certain F and f.

Algorithm CONSTRUCT_INSTANCE:

INPUT: Let F be a set of restricted FDs holding in a set of relations S with the same set of attributes U in database DB, and suppose f, $R\ (W, X \circledR Y)$, cannot be inferred from Axiom 4.1, where $R \subseteq S$, W is a set of restriction attributes, and X and Y are 2 sets of regular attributes from U.

OUTPUT: An instance of S in which the dependencies of F all hold, while f does not.

METHOD:

1 Find two relations $R1, R2 \in R$, such that Y is not a subset of $X^{+}_{(\{R1, R2\}, W)}$; /* $R1$ and $R2$ may refer to the same relation if R comprises a single relation.*/
// In what follows, we'll construct a tuple for each relation of $R1$ and $R2$. The other relations of S are empty.
2 Initialize each attribute in $R1$ and $R2$ as "null" value.
3 $RA := W$; // RA is the set of restriction attributes currently.
4 $XC := X^{+}_{(\{R1, R2\}, RA)}$;
5 **While** $\exists\ A \in U$, $R1[A] = R2[A] =$ null or ($A \in XC$ and $R1[A] \neq R2[A]$) **do**
6 **If** $A \in XC$ and ($R1[A] \neq R2[A]$ or $R1[A] = R2[A] =$ null) **then**
7 $R1[A] := a$ and $R2[A] := a$ for $a \in$ *dom(A)*, $A_{\sigma=\{a\}} \prec RA[A]$, and Y is not a subset of $X^{+}_{(\{R1, R2\}, RA \cup \{A_{\sigma=\{a\}}\})}$;
8 $RA := RA \cup \{A_{\sigma=\{a\}}\}$; /* Note if RA contains a restriction attribute $A_{\sigma=S}$, then the restriction attribute is replaced by $A_{\sigma=\{a\}}$ in the new RA.*/
9 **Else** // $A \notin XC$ and $R1[A] = R2[A] =$ null

10 $R1[A] := a1$ and $R2[A] := a2$ for $a1, a2 \in dom(A)$, $A_{\sigma=\{a1, a2\}} \prec RA[A]$, and Y is not a subset of $X^+_{(\{R1, R2\}, RA \cup \{A_{\sigma=\{a1,a2\}}\})}$;

11 $RA := RA \cup \{A_{\sigma=\{a1, a2\}}\}$;

12 $XC := X^+_{(\{R1, R2\}, RA)}$;

13 **Return** $\{R1, R2\}$; //Note the other relations of S are all empty. □

In what follows, we'll prove that Alg. CONSTRUCT_INSTANCE correctly constructs a set of relations in which the restricted FDs of F all hold but f does not. But before that, we need to prove Lemma 4.2 first.

Lemma 4.2: Let RA' and XC' be the values of variables of RA and XC after the execution of the algorithm. In the returned relations $R1$ and $R2$, all the attributes in XC' have the same non-null values, while the other attributes have different values.

Proof: First we prove that at the beginning of the k-th iteration of the while loop (Line 5~12), those attributes with the same non-null values in $R1$ and $R2$ belong to XC.

Base case: k=1. the condition holds as all attributes have null values.

Induction step: Let RA_k, XC_k be the value of variables RA and XC at the beginning of the k-th iteration. Suppose at the beginning of the k-th iteration, the condition holds. We have $RA_{k+1} \prec RA_k$, so $XC_{k+1} \supseteq XC_k$ (Lemma 4.1). For each attribute A with the same value in $R1$ and $R2$, if it is in XC_k, it is also in XC_{k+1}; otherwise, according to the induction hypothesis, the algorithm assigns the same value to A in $R1$ and $R2$ during the k-th iteration (Line 7). So $A \in XC_{k+1}$ according to the condition of Line 6.

So when the loop (Line 5~12) terminates, we have: all the attributes in $R1$ and $R2$ have non-null values, all the attributes of XC' have the same values in $R1$ and $R2$ (the condition of the while loop), and all the attributes with the same values in $R1$ and $R2$ belong to XC' (loop invariant proven above). That is, the attributes not in XC' have different values in $R1$ and $R2$. Finally, the while loop will terminate as in each iteration the algorithm assigns values to one attribute, and it will assign values to one attribute at most twice. □

Lemma 4.3: Given the input of Alg. CONSTRUCT_ INSTANCE, it constructs an instance of S in which the restricted FDs of F all hold but f doest not.

Proof: (1) The restricted FDs of F all hold in the set of relations $\{R1, R2\}$ returned by the algorithm.

Let RA' and XC' be the values of variables of RA and XC after the execution of the algorithm. We have proven that in the returned relations $R1$ and $R2$, all the attributes in XC' have the same values, while the others have different values.

Suppose a restricted FD $R1(W1, X1 \rightarrow Y1)$ in F does not hold in $\{R1, R2\}$, where $W1$ is a set of restriction attributes, and $X1$ and $Y1$ are sets of regular attributes. That is, $\{R1, R2\} \subseteq R1$, $RA' \prec W1 \cup X1$, and $Y1$ cannot be a subset of XC'. As the attributes of $X1$ have the same values in $R1$ and $R2$, $X1 \subseteq XC' = X^+_{(\{R1, R2\}, RA')}$ (Lemma 4.2). That is, $\{R1, R2\}(RA', X \rightarrow X1)$. Then we can derive $\{R1, R2\}(RA', X \rightarrow Y1)$. But then by the definition of attribute closure, $Y1$ is a subset of XC', which we assumed not to be the case. We conclude by contradiction that each restricted FD $R1(W1, X1 \rightarrow Y1)$ in F is satisfied by $\{R1, R2\}$.

(2) f: $R(W, X \rightarrow Y)$ does not hold in $\{R1, R2\}$.

We first try to prove that at the beginning of k-th iteration, $RA \prec W$ and Y is not a subset of XC.

Base case: k=1. $RA \prec W$ as $RA=W$. We can always find two relations $R1, R2 \in R$, such that Y is not a subset of $XC = X^+_{(\{R1, R2\}, RA)}$; otherwise, $\{Ri, Rj\}(W, X \rightarrow Y)$ can be deduced from F by Axiom 4.1 for any $Ri, Rj \in R$, then we can infer $R(W, X \rightarrow Y)$ by Axiom 4.1, rule (2). It contradicts the assumption that f cannot be deduced by Axiom 4.1.

Induction step: Let RA_k, XC_k be the value of variables RA and XC at the beginning of the k-th iteration. Suppose at the beginning of k-th iteration, the conditions hold. Let $RA_{k+1} = RA_k \cup \{A_{\sigma=S}\}$, then $RA_{k+1} \prec RA_k$. According to the induction hypothesis, we have $RA_k \prec W$. So $RA_{k+1} \prec W$. In what follows, we'll prove Y is not a subset of XC_{k+1} by contradiction. According to the induction hypothesis, $\{R1, R2\}(RA_k, X \rightarrow Y)$ cannot be deduced from F by Axiom 4.1. We consider two cases corresponding to the conditions of Line 6 and 9:

Case 1: $A \in XC_k$ and $RA_{k+1} = RA_k \cup \{A_{\sigma=\{a\}}\}$. We can always find a value a of attribute A, such that $\{R1, R2\}(RA_{k+1}, X \rightarrow Y)$ cannot be deduced from F by Axiom 4.1; otherwise, $\{R1, R2\}(RA_k, A_{\sigma=\{a_i\}}, X \rightarrow Y)$ can be deduced by Axiom 4.1 for each a_i such that $A_{\sigma=\{a_i\}} \prec RA_k[A]$. So $\{R1, R2\}(RA_k, A, X \rightarrow Y)$ (Axiom 4.1, rule (6)). As $A \in XC_k$, i.e., $\{R1, R2\}(RA_k, X \rightarrow A)$ holds, we can deduce $\{R1, R2\}(RA_k, X \rightarrow Y)$, which we assumed not to be the case.

Case 2: $A \notin XC_k$ and $RA_{k+1} = RA_k \cup \{A_{\sigma = \{a1,\ a2\}}\}$. We can always find two values *a1, a2* of attribute *A*, such that $\{R1,R2\}(RA_{k+1}, X \circledR Y)$ cannot be deduced from *F* by Axiom 4.1; otherwise, $\{R1, R2\}(RA_k, A_{\sigma = \{a_i,\ a_j\}}, X \circledR Y)$ can be deduced by Axiom 4.1 for any a_i, a_j such that $A_{\sigma = \{a_i,\ a_j\}} \prec RA_k[A]$. So $\{R1, R2\}(RA_k, X \circledR Y)$ (Axiom 4.1, rule (4)), which we assumed not to be the case.

When the loop terminates, $RA' \prec W$ and *Y* is not a subset of *XC'*. So there must be an attribute in *Y* which has different values in *R1* and *R2*. As $X \subseteq XC'$, all the attributes of *X* have the same values in *R1* and *R2*. Consequently, $R(W, X \circledR Y)$ does not hold in $\{R1, R2\}$. Note the loop will terminate as mentioned in the proof of Lemma 4.2. □

Consequently, we have the following theorem:

Theorem 4.1: The inference rules of Axiom 4.1 are sound and complete. □

4.3 Compute Attribute Closures With Respect To a Set of Restricted FDs

In this sub-section, we'll give an algorithm to compute attribute closure w.r.t a set of restricted FDs (see Section 4.2.1 for the definition of attribute closure). We require certain conditions on the given set of restricted FDs to simplify the following CLOSURE algorithm. The reason why we require those conditions on the input restricted FDs will be explained in Section 5.3. The restricted FDs we consider capture a wide range of FD-like ICs in practice.

Algorithm CLOSURE (*S, U, R, W, X, F*)

INPUT: A set of relations *S* with the same set of attributes *U*;
A set of relation names $R \subseteq S$;
A set of restriction attributes *W* from *U*;
A set of regular attributes *X* from *U*;
F: a set of restricted FDs satisfies: (1) the restriction attributes of any restricted FD of *F* are restricted to take single values, i.e., any restriction attribute has a form of $A_{\sigma = \{a\}}$, and (2) let *Z1* be the set of regular attributes, and *Z2* be the set of attributes occurred in some restriction attributes of the restricted FDs of *F*. Then $Z1 \cap Z2 = \emptyset$. (3) Each restricted FD of *F* holds either in a single relation, or in a set of relations $R_B = dom(B)$ for some attribute *B* whose values are relation names in *S*.

OUTPUT: $X^+_{(R, W)}$ w.r.t. *F*

METHOD: Let $\{X1, ..., Xn\}$ be a subset of *X* such that each *Xi, i=1, ..., n*, occurs in some restriction attribute of the restricted FDs of *F*.

1 *closure* := *U;*
2 **For** any $x1 \in dom(X1), \ldots, xn \in dom(Xn)$, such that $\{X1_{\sigma = \{x1\}}, \ldots, Xn_{\sigma = \{xn\}}\} \prec W[X1, ..., Xn]$ **do**
3 $closure1 := X \cup \{A \mid A_{\sigma = \{a\}} \in W\}$;
4 $W1 := W \cup \{X1_{\sigma = \{x1\}}, \ldots, Xn_{\sigma = \{xn\}}\}$;
5 **Repeat** until no change on *closure1*:
6 **If** there's a restricted FD $R1(\overline{Y} \circledR Z)$ in *F* such that $R \subseteq R1$ and $W1 \cup closure1 \prec \overline{Y}$, **Then**
7 $closure1 := closure1 \cup Z$;
8 $closure := closure \cap closure1$;
9 **Return** *closure;* □

Example 4.2: In *DB1* of Figure 1.1, given a set of restricted FDs: $Supply(product, supplier, month_{\sigma=\{mi\}} \circledR price)$ for each $mi \in \{Jan, ..., Dec\}$, let $X=\{product, supplier, month\}$, we want to compute the attribute closure of *X* without any restriction in the *Supply* relation, i.e., $X^+_{(Supplier, \emptyset)}$ using the CLOSURE algorithm. For each $mi \in \{Jan, ..., Dec\}$, let $W1 = \{month_{\sigma=\{mi\}}\}$, then after each iteration of inner loop (Line 5~7), $closure1 = X^+_{(Supplier, W1)} = \{product, supplier, month, price\}$. Consequently, the algorithm returns $closure = X^+_{(Supplier, \emptyset)} = \{product, supplier, month, price\}$.□

In the rest of this sub-section, we prove the correctness of the above algorithm.

Lemma 4.4: Given any $x1 \in dom(X1), \ldots, xn \in dom(Xn)$, such that $\{X1_{\sigma=\{x1\}}, \ldots, Xn_{\sigma=\{xn\}}\} \prec W[X1, ..., Xn]$, let $W1 = W \cup \{X1_{\sigma=\{x1\}}, \ldots, Xn_{\sigma=\{xn\}}\}$, the inner loop (Line 5~7) correctly computes *closure1*, $X^+_{(R, W1)}$ w.r.t *F*.

Proof: First if *A* is in the set *closure1* produced by the inner loop, then $A \in X^+_{(R, W1)}$. This can be proven easily by induction. We now prove the converse: if $A \in X^+_{(R, W1)}$, then *A* is in *closure1* produced by the inner loop. Suppose $A \in X^+_{(R, W1)}$, but *A* is not in *closure1* produced by the inner loop.

Consider 2 relations $R1, R2 \in R$ ($R1$ and $R2$ may refer to the same relation if R comprises only relation). Each of $R1$ and $R2$ has a tuple that agree on the attributes of *closure1* and disagree on all the other attributes, and the attribute values satisfy the restriction of $W1$. We claim $\{R1, R2\}$ satisfies F. If not, let $R1(V, Y \circledR Z)$ (V is a set of restriction attributes and Y and Z are 2 sets of regular attributes from U) be a dependency of F that is violated by $\{R1, R2\}$. That is, $\{R1, R2\} \subseteq R1$ and $\{A_{\sigma = R1[A] \cup R2[A]} \mid A \in U\} \prec V \cup Y$, and Z cannot be a subset of *closure1*.

We claim that $V \subseteq \{X1_{\sigma = \{x1\}}, \ldots, Xn_{\sigma = \{xn\}}\} \cup \{A_{\sigma = \{a\}} \mid A_{\sigma = \{a\}} \in W\}$. First as the restriction attributes of V are restricted to take single values under our assumption, the attributes of V are all from *closure1*; otherwise, an attribute of V has different values in $R1$ and $R2$, and the attribute values of $R1$ and $R2$ cannot satisfy V. Second the attributes of V will not be on the right hand side of a restricted FD under our assumption that there're no intersection between the sets of regular and restriction attributes of the restricted FDs of F, so the attributes of V will not be added into *closure1* during the inner loop (Line 5~7). So all the attributes of V can only be added into *closure1* on Line 3, i.e., $V \subseteq \{X1_{\sigma = \{x1\}}, \ldots, Xn_{\sigma = \{xn\}}\} \cup \{A_{\sigma = \{a\}} \mid A_{\sigma = \{a\}} \in W\}$.

So $W1 \prec V$. As $Y \subseteq closure1$, $closure1 \prec Y$. So $W1 \cup closure1 \prec V \cup Y$. So $Z \subseteq clsoure1$ (Line 6, 7). We conclude by contradiction that each restricted FD of F holds in $\{R1, R2\}$.

Thus, the relation set $\{R1, R2\}$ must also satisfy $R(W1, X \circledR A)$. The reason is that we assume $A \in X^+_{(R, W1)}$, i.e., $R(W1, X \circledR A)$ follows from F by Axiom 4.1. Since the axiom is sound, any relation set satisfying F satisfies $R(W1, X \circledR A)$. But the only way $R(W1, X \circledR A)$ could hold in $\{R1, R2\}$ is if A is in *closure1*, for if not, then the attribute values of $R1$ and $R2$ which satisfy $W1 \cup X$, would disagree on A and violate $R(W1, X \circledR A)$. □

Theorem 4.2: Algorithm CLOSURE correctly computes $X^+_{(R, W)}$ w.r.t. F.

Proof: For any $x1 \in dom(X1), \ldots, xn \in dom(Xn)$, such that $\{X1_{\sigma = \{x1\}}, \ldots, Xn_{\sigma = \{xn\}}\} \prec W[X1, \ldots, Xn]$, let $W1 = W \cup \{X1_{\sigma = \{x1\}}, \ldots, Xn_{\sigma = \{xn\}}\}$. If $A \in closure$, then $A \in X^+_{(R, W1)}$, i.e., $R(W1, X \circledR A)$ holds for each $W1$. So $R(W, X \circledR A)$ can be deduced by Axiom 4.1. That is, $A \in X^+_{(R, W)}$.

On the other hand, if $A \in X^+_{(R, W)}$, then $A \in X^+_{(R, W1)}$ for each $W1$. According to Lemma 4.3, A will be returned by each iteration of the inner loop on Line 5~7. So A will be in the set *closure*. □

As to the complexity of Alg. CLOSURE, we have the following lemma.

Lemma 4.5: Given the input of Alg. CLOSURE, let Z be the set of attributes occurred in some restriction attributes of the restricted FDs of F. Let m be the cardinality of Z, and d be an upper bound for the cardinalities of the domains of the attributes of Z. Then the algorithm takes time $O(d^m |U| |F|^2)$.

Proof: The outer loop of the CLOSURE algorithm (Line 2~8) will be iterated for d^m times at most. For each iteration of the outer loop, the inner loop (Line 5~7) would be repeated for $O(|F|^2)$ times, and each iteration of the inner loop takes $O(|U|)$ time. Consequently, the whole algorithm takes $O(d^m |U| |F|^2)$ time. □

From Lemma, 4.5, we know that the performance of Alg. CLOSURE depends much on the structure of the set Z. When the parameters m and d are constants, the algorithm runs in polynomial time. In Section 5.4, we'll study the complexity of the CLOSURE algorithm in more details in the context of schematic discrepant transformation.

4.4 Impact of *no-value*

In the definitions of *fold* and *unfold* (Section 2.1), we introduced *no-value* when some attribute is inapplicable for a tuple. *no-value* does not belong to the domain of any attribute. It is also different from *null* value which could be *unknown* or *no-value*. In practice, ICs of restricted FDs are usually specified when involved attributes take domain values; *no-value* will not cause the violation of a restricted FD. To check whether a restricted FD holds on a relation (or a set of relations), we need only check those tuples with domain values on the attributes involved in the restricted FD. However *no-value* makes some difference to the inference rules of restricted FDs. Specifically, if attributes can take *no-value*, the rule (9) of Axiom 4.1 should be:

If $R(\overline{X} \circledR Y)$ and $R(\overline{X1}, Y \circledR Z)$ hold, where $\overline{X1}$ is a set (possibly an empty set) of some restriction attributes in $\overline{X}$, let Y_i, $i=1, \ldots, m$, be all the attributes in Y which could take *no-value* [3], then

$$R(Y_{1\,\sigma=dom(Y_1)}, \ldots, Y_{m\,\sigma=dom(Y_m)}, \overline{X} \circledR Z) \text{ holds.}$$

[3] Under our assumption, Y_i's are attributes created by *unfold* operators.

In the above inferred restricted FD, those restriction attributes $Y_{i\,\sigma=dom(Y_i)}$ mean that the restricted FD holds over tuples in which those attributes Y_i's have domain values. On the other hand, for those tuples in which some Y_i's take *no-value*, $R(\overline{X} \rightarrow Y)$ and $R(\overline{X1}, Y \rightarrow Z)$ hold trivially no matter what values the attributes in $\overline{X}$ and Z take, and therefore $R(\overline{X} \rightarrow Z)$ may not hold. Specifically, if all the attributes of Y cannot take *no-value*, then the inferred restricted FD is just: $R(\overline{X} \rightarrow Z)$.

To simplify the presentation, in the rest of the paper, we assume all attributes will only take domain values. However our results can be extended to handle *no-value* easily.

5 Propagation of Restricted FDs in Schematic Discrepant Transformations

In Section 4.2, the inference rules allow us to infer unknown restricted FDs from known ones in a fixed relation (or a relation set). In this section, we'll give another kind of inference rules (called ***propagation rules***) which allow us to infer restricted FDs of transformed relations from restricted FDs of original relations in a transformation.

In general, given a transformation T, let R_0 and R_k be the sets of original and target transformed relations of T; let F be a set of restricted FDs for the set of relation schemas of R_0, and let f be a restricted FD for the set of relation schemas of R_k; let r_0 be any instance of R_0 satisfying the dependencies in F, and let r_k be the instance of R_k transformed from r_0 by T. We say F (*logically*) *implies* f, if r_k satisfies f.

We extend the definition of the equivalences among sets of restricted FDs for discrepant schemas: given a transformation T, let F and G be two sets of restricted FDs holding in the original and target transformed relations of T. We say F and G are ***equivalent*** if F implies each restricted FD in G, and vice versa.

Given a transformation T, let R_0 and R_k be the sets of original and target transformed relations of T; let F be a set of restricted FDs for the set of relation schemas of R_0, and G be the set of restricted FDs for R_k implied by F. We call T *a restricted-FD preserving transformation* if F is equivalent to G.

In general, schematic discrepant transformations are not restricted-FD preserving transformations. Our purpose is to compute all the *reasonable restricted FDs* (will be defined in Section 5.2) in transformed relations implied by restricted FDs in original relations. Section 5.1 studies how restricted FDs change in application of each kind of restructuring operators. Section 5.2 gives algorithms to deduce restricted FDs in a transformation, i.e., a sequence of restructuring operators. Section 5.3 proves the completeness of our algorithm. Section 5.4 analyzes the time complexity of the algorithm.

5.1 Propagation Rules of Restricted FDs in Application of Restructuring Operators

In this sub-section, we study how restricted FDs change in application of each kind of restructuring operators. We study the propagation rules for *split/unite* and *unfold/fold* operators in a pairwise way. The proof of the soundness of these rules is easy and suppressed; the completeness will be described in Theorem 5.1 (Section 5.2.1).

Axiom 5.1 (*Propagation rule of restricted FDs in application of a split/unite operator*):

Let $R(A_1, ..., A_n, B)$ be an original relation of *DB1*, and $b_i(A_1, ..., A_n)$ ($i = 1, ..., m$) be the transformed relations of *DB2* using *split(R, B)*, i.e., the distinct values of B in R, $\{b_1, ..., b_m\}$, become relation names of the transformed relations. Let $\overline{X}$ be a mixed set of regular and restriction attributes from $\{A_1, ..., A_n\}$, and $Y \subseteq \{A_1, ..., A_n\}$ be a set of regular attributes; let $R_B \subseteq \{b_1, ..., b_m\}$. We have the following rule:

$$R(B_{\sigma=R_B}, \overline{X} \rightarrow Y) \text{ holds in } DB1 \text{ iff } R_B(\overline{X} \rightarrow Y) \text{ holds in } DB2.$$

The same rule holds for *unite* operator. That is, let b_i, $i = 1, ..., m$, be the original relations, and R be the transformed relation using *unite*($\{b_1, ..., b_m\}$, B). Then the above rule holds.□

For example, in Example 1.2, we can infer the restricted FD in *DB3* from the one in *DB1* (actually the equivalent restricted FD derived in Example 4.1) and vice versa, or infer the restricted FD in *DB4* from the one in *DB2*, and vice versa using Axiom 5.1.

unite is a restricted-FD preserving transformation. This is described as the following lemma, the proof of which is easy and suppressed.

Lemma 5.1: In application of *unite*, the rule in Axiom 5.1 change any restricted FD in the original relations into an equivalent restricted FD in the transformed relation. □

Although *unite* is a restricted-FD preserving transformation, *split* is not. Given the original and transformed relations for *split* operator in Axiom 5.1, a restricted FD $R(\overline{X} \rightarrow B)$ will be lost in application of *split*, as the values of B become relation names in the transformed relations.

In what follows, we'll give rules on the changes of restricted FDs in application of a set of *unfold/fold* operators. We study based on a set of *unfold/fold* operators (e.g., in Example 2.2, transform *DB5* into *DB4* with a set of *unfold* operators) instead of individual operators because some restricted FDs would hold over a set of relations instead of individual relations.

Axiom 5.2 (*Propagation rules of restricted FDs in application of a set of unfold/fold operators*):
Let $R_i(A_1, ..., A_n, B, C)$, $i=1,...,l$, be original relations of *DB1* with FD $A_1, ..., A_n, B \to C$ holding in each relation $DB1::R_i$, and $R_i(A_1, ..., A_n, b_1, ..., b_m)$ be the transformed relation of *DB2* using $unfold(DB1::R_i, B, C)$ for each $i = 1,...,l$, i.e., the values of B in $DB1::R_i$, $b_1, ..., b_m$, become attribute names in $DB2::R_i$, and the values of C in $DB1::R_i$ become the values of attributes $b_1, ..., b_m$ in $DB2::R_i$. Let $\overline{X}$ be a mixed set of regular and restriction attributes from $\{A_1, ..., A_n\}$, and $Y \subseteq \{A_1, ..., A_n\}$ be a set of regular attributes. Let $R^{(1)}$ ($R^{(2)}$) be a subset of relations of $\{R_1, ..., R_l\}$ from *DB1* (*DB2*), and $R^{(1)}$ and $R^{(2)}$ contain the same set of relation names. We have the following rules:

(1) $R^{(1)}(B_{\sigma=\{b_i\}}, \overline{X} \to C)$ holds iff $R^{(2)}(\overline{X} \to b_i)$ holds.
(2) $R^{(1)}(B_{\sigma=\{b_i\}}, \overline{X}, C \to Y)$ holds iff $R^{(2)}(\overline{X}, b_i \to Y)$ holds.
(3) $R^{(1)}(B_{\sigma=\{b_i\}}, \overline{X} \to Y)$ holds iff $R^{(2)}(b_{i\ \sigma=dom(b_i)}, \overline{X} \to Y)$ holds.
(4) $R^{(1)}(\overline{X} \to Y)$ holds iff $R^{(2)}(\overline{X} \to Y)$ holds.

The 4 rules also hold for *fold* operators. That is, let $DB2::R_i$, $i = 1, ...,l$, be original relations with FD $A_1, ..., A_n \to b_1, ..., b_m$ holding in each relation $DB2::R_i$, and $DB1::R_i$ be the transformed relations using $fold(DB2::R_i, B, C)$. Then the 4 rules hold. □

For example, in Example 1.2, using rule (1), we can infer the restricted FD in *DB2* from the one in *DB1* (actually the equivalent restricted FD derived in Example 4.1), and vice versa.

A *fold* transformation preserves restricted FDs with certain forms (see the following lemma, the proof of which is suppressed).

Lemma 5.2: Let $DB1::R_i(A_1, ..., A_n, b_1, ..., b_m)$, $i=1,...,l$, be original relations with FD $A_1, ..., A_n \to b_1, ..., b_m$ holding in each relation $DB1::R_i$, and $DB2::R_i(A_1, ..., A_n, B, C)$ be the transformed relation using $fold(DB1::R_i, B, C)$ for each $i = 1,...,l$. Given a set of restricted FDs F in *DB1* such that each restricted FD of F contains at most one b_i ($1 \le i \le l$) as regular attribute, the rules in Axiom 5.2 changes each restricted FD of F into an equivalent one in *DB2*.□

5.2 Inferring Restricted FDs in Schematic Discrepant Transformations

In this subsection, we give 2 algorithms to infer restricted FDs in a transformation consisting of a sequence of restructuring operators.

General restricted FDs are powerful to express a broad class of ICs, which cause the inference and propagation of restricted FDs to take much time. To simplify the inference and propagation of restricted FDs and focus on ICs which are popular in practice, we consider a special class of restricted FDs, called ***reasonable restricted FDs***, defined below.

Definition 5.1: Let S be a set of original relations with the same set of attributes U in database *DB*; let F be the set of restricted FDs holding in S. We call a restricted FD f: $R(X \to Y)$ from F ***reasonable*** if it satisfies 3 conditions:
(1) Either $R = dom(B)$ for some attribute B whose values are modeled as relation names in S, or R is a single relation name from S.
(2) The restricted FD only has regular attributes.
(3) For each attribute set $Z = \{b_i \mid b_i \in U$ is a value of some 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 $X \cup Y$. □

Now the problem becomes: given a transformation and a set of reasonable restricted FDs holding in original relations of the transformation, compute all the reasonable restricted FDs holding in the target transformation relations. Algorithm NAIVE_PROPAGATE (Section 5.2.1) is an exponential-time algorithm. We give it because: (1) it describes a general process to infer restricted FDs in a transformation, and (2) the completeness of the rules in Axiom 5.1 and 5.2 is proven based on the naive algorithm. Then in Section 5.2.2, we'll give a more efficient algorithm.

5.2.1 A Naive Algorithm

We first give the algorithm, then prove the soundness and completeness of it.

Algorithm NAIVE_PROPAGATE**:**

INPUT: (1) a transformation, T, consisting of a sequence of restructuring operators, $<T_1, ..., T_k>$; (2) a set of original relations of T, R_0; (3) a set of reasonable restricted FDs, F_0, holding in the relations of R_0;
OUTPUT: a set of reasonable restricted FDs, F_k, holding in the target transformed relations, R_k.
METHOD: For each operator (or operator set) T_i, $i = 1, ..., k$, let R_{i-1} and R_i be the original and transformed relations of T_i, and F_{i-1} be the set of restricted FDs holding in R_{i-1}. We compute the set of restricted FDs, F_i, holding in R_i in 2 steps:

1. Compute F_{i-1}^+ from F_{i-1} using Axiom 4.1;
2. Compute F_i from F_{i-1}^+ using Axiom 5.1 or 5.2.

Finally, we get the set of restricted FDs, F_k, holding in the target transformed relations of R_k. □

In what follows, we'll prove the soundness and completeness of Alg. NAIVE_PROPAGATE.

Lemma 5.3: Given a transformation T consisting of *unite* and *fold* operators, for each reasonable restricted FD f holding in the original relations of T, we can infer an equivalent restricted FD in the target transformed relations using Alg. NAIVE_PROPAGATE.

Proof: This can be concluded from Lemma 5.1 and 5.2 readily. □

Given the input of Algorithm NAIVE_PROPAGATE, it correctly computes F_k whose closure includes all the reasonable restricted FDs holding in R_k. We can prove the completeness of Algorithm NAIVE_PROPAGATE by showing that if a reasonable restricted FD $f \notin F_k^+$, then there must be an instance r_0 of R_0 in which the dependencies of F_0 all hold but in the target transformed relations of r_k, f doest not hold. That is, F_0 does not logically imply f. The result is given below.

Theorem 5.1: Algorithm NAIVE_PROPAGATE is sound and complete to infer reasonable restricted FDs in a lossless transformation.

Proof: The algorithm is sound because each restricted FD computed by it holds in R_k. This can be concluded from the soundness of the rules of Axiom 4.1, 5.1 and 5.2. In what follows, we prove the completeness.

Without loss of generality, we assume the given lossless transformation, T, is non-redundant. Recall such a transformation T can be implemented in two phases (see Section 2): (1) transform the original relation set R_0 into R_i with *unite* and *fold* operators, such that schema labels in R_0 become attribute values in R_i; (2) transform R_i into the target transformed relation set R_k using *unfold* and *split* operators.

Suppose a reasonable restricted FD f for R_k cannot be derived from F_0 using Alg. NAIVE_PROPAGATE. According to Lemma 5.3, we can infer an equivalent restricted FD g of f in R_i (Note R_i can be transformed from R_k using *unite* and *fold* operators as T is a lossless transformation). We can also infer a set of restricted FDs F_i for R_i which are equivalent to F_0 according to Lemma 5.3

We claim F_i does not imply g. This can be proven by contradiction. Suppose F_i implies g. As the rules of Axiom 4.1 are complete, we can infer g from F_i by Axiom 4.1. As we can infer F_i from F_0, and f from g, we can infer f from F_0 by Alg. NAIVE_PROPAGATE. This contradicts our assumption.

Consequently, there exists an instance r_i of R_i, such that the restricted FDs in F_i all hold but g does not. Let r_0 and r_k be the instances of original and target transformed relations of r_i. Then F_0 holds in r_0 as F_i holds in r_i. However, f does not hold in r_k as g does not hold in r_i. □

5.2.2 A More Efficient Algorithm

Algorithm NAIVE_PROPAGATE needs to compute restricted FD closures, whose size may be exponential in the number of given restricted FDs. We develop a more efficient and still complete algorithm to infer restricted FDs in a transformation.

In Algorithm NAIVE_PROPAGATE, for each restructuring operator (or operator set) T_i, $i = 1, ..., k$, we first compute F_{i-1}^+ from F_{i-1}, then F_i from F_{i-1}^+. We'll improve the algorithm by computing F_i from F_{i-1} directly, without computing the restricted FD closure of F_{i-1}. To this end, we can divide the restricted FDs of F_{i-1} into 2 groups. The restricted FDs of group 1 can be changed into equivalent restricted FDs in F_i, i.e., those restricted FDs can be recovered when applying a converse transformation of T_i; the restricted FDs of group 2 cannot be preserved in F_i, i.e., those restricted FDs are either changed into a weaker restricted FD or cannot be changed into any restricted FDs in F_i. For group 1, we only need to change the restricted FDs in that group, and needn't consider the other restricted FDs implied by them, as those implied ones are

preserved naturally. For group 2, we need consider not only the restricted FDs in that group, but also those implied ones which are not in F_{i-1} but in F_{i-1}^+ and can be changed into some restricted FDs in F_i.

In what follows, we'll give a sub-algorithm to infer restricted FDs without computing restricted FD closures in application of *split* operators. The sub-algorithms for the other operators will be given in Appendix. The main process which calls these sub-algorithms to infer restricted FDs in a transformation is similar to Algorithm NAIVE_PROPAGATE: Given a transformation $<T_1, ..., T_k>$ and a set of reasonable restricted FDs in the original relations, call a sub-algorithm for each operator (or operator set) T_i, $i = 1, ..., k$, in sequence, and finally get the restricted FDs in the target transformed relations. For easy to reference, we call this main process **Algorithm EFFICIENT_PROPAGATE** which is omitted by us.

To simplify the presentation, in the following sub-algorithm and those in Appendix, we assume each input restricted FD has single attribute on the right. Although we require the input restricted FDs of Algorithm EFFICIENT_PROPAGATE are reasonable and therefore only have regular attributes, the input restricted FDs of a sub-algorithm may have restriction attributes, as a restricted FD without restriction attributes may be changed into one with restriction attributes during *unite* and *fold* transformations.

Algorithm INFER_SPLIT: *Inference of restricted FDs for split operator.*

INPUT: Let $R(A_1, ..., A_n, B)$ be an original relation of *DB1*, and $b_i(A_1, ..., A_n)$, $i = 1, ..., m$, be the transformed relations of *DB2* using *split(R, B)*.
Let F be a set (not necessary a closure) of restricted FDs in *DB1::R*.

OUTPUT: a set of restricted FDs, G, holding in the set of transformed relations $\{b_1, ..., b_m\}$ of *DB2*.

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

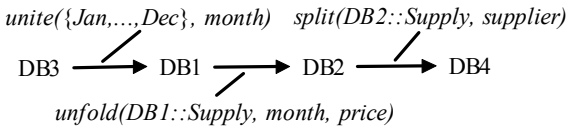
(1) If $R(\overline{X} \to A) \in F$, then $dom(B)(\overline{X} \to A) \in G$.
(2) If $R(\overline{X}, B \to A) \in F$, then $b_i(\overline{X} \to A) \in G$ for each $b_i \in dom(B)$.
(3) If $R(\overline{X} \to B) \in F$ and $R(\overline{Y}, B \to A) \in F$, then $dom(B)(\overline{X}, \overline{Y} \to A) \in G$. □

The 3 rules in the above algorithm can be proven easily using Axiom 4.1 and Axiom 5.1. In rule (3), although $R(\overline{X} \to B)$ cannot be changed into any restricted FD in *DB2*, it and $R(\overline{Y}, B \to A)$ together imply $R(\overline{X}, \overline{Y} \to A)$ (note B cannot take *no-value* under our assumption) which is changed into $dom(B)(\overline{X}, \overline{Y} \to A)$ in *DB2*.

Unlike Algorithm NAIVE_PROPAGATE, the rules of the above algorithm apply to the restricted FDs of F directly, without computing the closure of F. This greatly reduces the time complexity. In worst case, Algorithm INFER_SPLIT takes time proportional to the square of the number of restricted FDs in F. The worst case occurs when half of the restricted FDs in F have a form of $R(\overline{X} \to B)$, and the other half have a form of $R(\overline{Y}, B \to A)$. According to rule (3), the size of G is proportional to the square of the size of F. Note the set of restricted FDs generated by rule (2) are alike in form, and can be presented and dealt with as one special dependency, which will not dominate the running time of the algorithm.

Similarly, we give the sub-algorithms for the other 3 operators in Appendix. In what follow, we give an example to infer restricted FDs in a transformation consisting of 3 operators.

Example 5.1: Suppose in Figure 1.1, we transform *DB3* into *DB4*:



Suppose in the relations of *DB3*, the following restricted FDs hold: $mi(product, supplier \to price)$ for each $mi \in \{Jan, ..., Dec\}$. In what follows, we compute the restricted FDs in *DB4* using Algorithm EFFICIENT_PROPAGATE:

After applying the *unite* operator, we get the dependencies on *DB1*:
$Supply(product, supplier, month_{\sigma=\{mi\}} \to price)$ for each $mi \in \{Jan, ..., Dec\}$. (Alg. INFER_UNITE, rule (2))

After applying the *unfold* operator, we get the dependencies on *DB2*:
$Supply(product, supplier \to mi)$ for each $mi \in \{Jan, ..., Dec\}$. (Alg. INFER_UNFOLD, rule (1))

After applying the *split* operator, we get the dependencies on *DB4*:
$si(product \to mi)$ for each $mi \in \{Jan, ..., Dec\}$ and each $si \in \{s1, ..., sn\}$. (Alg. INFER_SPLIT, rule (2)) □

5.3 Completeness of Alg. EFFICIENT_PROPAGATE

In this sub-section, we'll prove the completeness of the EFFICIENT_PROPAGATE algorithm. As mentioned, during a transformation, a reasonable restricted FD may become a restricted FD with restriction attributes. The following lemma describes the form of restricted FDs generated during the application of Alg. EFFICIENT_PROPAGATE.

Lemma 5.4: Suppose we are given a transformation and a set of reasonable restricted FDs for the original relations of the transformation as an input to Alg. EFFICIENT_PROPAGATE. Then the set of restricted FDs F produced by a sub-algorithm (INFER_SPLIT, INFER_UNITE, INFER_UNFOLD or INFER_FOLD) satisfies the following 4 conditions:

(1) The restriction attributes of any restricted FD of F are restricted to take single values.

(2) Let $Z1$ be the set of regular attributes, and $Z2$ be the set of attributes occurred in some restriction attributes of the restricted FDs of F. Then $Z1 \cap Z2 = \emptyset$.

(3) Condition (1) of Definition 5.1.

(4) Condition (3) of Definition 5.1.□

The lemma can be proven by induction. Note the first 3 conditions given here are the same as those on the input restricted FDs to Algorithm CLOUSRE (Section 4.3).

Definition 5.2: A derivation sequence s for a restricted FD $R(\overline{X} \rightarrow A)$ is a sequence of restricted FDs from a set of restricted FDs F:

$$R1(\overline{Y1} \rightarrow A1), \ldots, Rm(\overline{Ym} \rightarrow Am)$$

such that $Ri \subseteq R$, $\overline{X} \prec \overline{Y1}$, $Am=A$, and $\overline{X} \cup \{A1, \ldots, Ai\text{-}1\} \prec \overline{Yi}$ for $i=1,\ldots,m$.□

Definition 5.3: We say that a derivation sequence s lies in a set of attributes U iff for each restricted FD f of s, all the (regular and restriction) attributes of f are from U. □

Definition 5.4: Given a set of restricted FDs F and a restricted FD $R(W, X \rightarrow A)$ with the restriction attribute set W and regular attribute set X, let $\{X1, \ldots, Xn\}$ be a subset of X such that each Xi, $i=1, \ldots, n$, occurs in some restriction attribute of the restricted FDs of F. A derivation sequence set S for $R(W, X \rightarrow A)$ is a set of derivation sequences from F:

S = {s | s is a derivation sequence for $R(W, X1_{\sigma=\{x1\}}, \ldots, Xn_{\sigma=\{xn\}}, X \rightarrow A)$ for any $x1 \in dom(X1), \ldots, xn \in dom(Xn)$ such that $\{X1_{\sigma=\{x1\}}, \ldots, Xn_{\sigma=\{xn\}}\} \prec W[X1, \ldots, Xn]$} □

Lemma 5.5: Given a set of restricted FDs F produced by a sub-algorithm of EFFICIENT_PROPAGATE and a restricted FD f, $f \in F^+$ iff f is a trivial restricted FD or there exists a derivation sequence set S for f.

Proof: Given f: $R(W, X \rightarrow A)$, we amend the CLOSURE algorithm to check whether $A \in X^+_{(R, W)}$. That is, for a value sequence $(x1, \ldots, xn)$ of $(X1, \ldots, Xn)$ such that $\{X1_{\sigma=\{x1\}}, \ldots, Xn_{\sigma=\{xn\}}\} \prec W[X1, \ldots, Xn]$, the inner loop (Line 5~7) of Alg. CLOSURE terminates when either attribute A occurs in the set *closure1* or no more attribute can be added to *closure1* (in the latter case, the whole algorithm returns reporting that $A \notin X^+_{(R, W)}$). For each iteration of inner loop, the sequence of restricted FDs chosen in Line 6 constitutes a derivation sequence for f. Finally, if we can find such a derivation sequence for each value sequence $(x1, \ldots, xn)$ satisfying the restriction of W, then $A \in X^+_{(R, W)}$ and $f \in F^+$. Consequently, this lemma follows easily from the correctness of Alg. CLOSURE (Theorem 4.2 in Section 4.3).□

We are ready to prove the completeness of Alg. EFFICIENT_PROPAGATE now. We'll prove this by showing that Alg. EFFICIENT_PROPAGATE generates the same result as Alg. NAIVE_PROPAGATE. In what follows, we first prove the completeness of the four sub-algorithms Alg. INFER_SPLIT, INFER_UNITE, INFER_UNFOLD and INFER_FOLD, then the completeness of Alg. EFFICIENT_PROPAGATE.

Lemma 5.6: Each of the four sub-algorithms INFER_SPLIT, INFER_UNITE, INFER_UNFOLD and INFER_FOLD is complete if the input restricted FDs satisfy the 4 conditions in Lemma 5.4.

Proof: We only prove the completeness of Alg. INFER_SPLIT. The proofs for the other sub-algorithms are similar.

Let $R(A_1, \ldots, A_n, B)$ be an original relation, and $b_i(A_1, \ldots, A_n)$, $i = 1, \ldots, m$, be the transformed relations using *split(R, B)*. Let F be a set (not necessary a closure) of restricted FDs in R. Let G and G' be the sets of restricted FDs computed by Alg. NAIVE_PROPAGATE and INFER_SPLIT resp. We'll prove G is equivalent to G'.

It is easy to see that $G' \subseteq G^+$ as the rules of Alg. INFER_SPLIT are derived from the inference rules (Axiom 4.1) and propagation rules (Axiom 5.1). We'll prove $G \subseteq G'^+$ as follows.

For each dependency g: $R_B(W, X \to A)$ in G with $R_B \subseteq \{b_1, ..., b_m\}$, W a set of restriction attribute and $X = \{X1, ..., Xn\}$ a set of regular attributes from $\{A_1, ..., A_n\}$, and $A \in \{A_1, ..., A_n\}$, let f: $R(B_{\sigma=R_B}, W, X \to A)$ be the original restricted FD in F^+. According to Lemma 5.5, there's a set of derivation sequence set S from F for f. Let s: $R(\overline{Y1} \to C1)$, …, $R(\overline{Ym} \to Cm)$ be a derivation sequence in S for the restricted FD: $R(B_{\sigma=R_B}, W, X1_{\sigma=\{x1\}}, …, Xn_{\sigma=\{xn\}} \to A)$ for any $x1 \in dom(X1), …, xn \in dom(Xn)$ and $\{X1_{\sigma=\{x1\}}, …, Xn_{\sigma=\{xn\}}\} \prec W[X1, ..., Xn]$. We'll construct a derivation sequence s' from G' for $R_B(W, X1_{\sigma=\{x1\}}, …, Xn_{\sigma=\{xn\}} \to A)$ in the transformed relations by transforming the restricted FDs of s.

For each i=1, …, m, let s_i and s_i' be the i-th restricted FD in s and s' resp., we consider 3 cases of s_i:

Case 1: If s_i: $R(\overline{Yi} \to Ci)$ lies in $\{A_1, ..., A_n\}$, then we derive s_i': $dom(B)(\overline{Yi} \to Ci)$ from s_i, using rule (1) of Alg. INFER_SPLIT.

Case 2: Either s_i contains $B_{\sigma=\{b\}}$ for some $b \in dom(B)$, or s_i contains B in its LHS but all s_j with $j < i$ does not contain B in their RHS. In this case, f must have a form of $R(B_{\sigma=\{b\}}, W, X \to A)$ (note in this case, g has a form of $b(W, X \to A)$); otherwise s_i would not be in the derivation sequence set of f as $\{W, X1_{\sigma=\{x1\}}, …, Xn_{\sigma=\{xn\}}\} \cup \{C1, ..., Ci\text{-}1\}$ does not satisfy $\overline{Yi}$. Then we derive s_i': $b(\overline{Yi}' \to Ci)$ such that $\overline{Yi} = \overline{Yi}' \cup \{B\}$ or $\overline{Yi} = \overline{Yi}' \cup \{B_{\sigma=\{b\}}\}$ from s_i, using rule (2) of Alg. INFER_SPLIT.

Case 3: s_i contains B on the RHS, i.e., $R(\overline{Yi} \to B)$. Without loss of generality, we can assume s contains exactly one such restricted FD (it would be no need to derive B more than once). We construct s' from s by applying rule (3) of Alg. INFER_SPLIT: for each j, with $i < j \le m$, if s_j contains B in its LHS then construct s_j': $dom(B)(\overline{Yi}, \overline{Yj}' \to Cj)$ such that $\overline{Yj} = \overline{Yj}' \cup \{B\}$. Note in this case, s_i itself would not be changed into any s_i'.

It is easy to see that s' is a derivation sequence for $R_B(W, X1_{\sigma=\{x1\}}, …, Xn_{\sigma=\{xn\}} \to A)$ from G'. It follows that $R_B(W, X1_{\sigma=\{x1\}}, …, Xn_{\sigma=\{xn\}} \to A) \in G'^+$. As this is true for any $x1 \in dom(X1), …, xn \in dom(Xn)$ such that $\{X1_{\sigma=\{x1\}}, …, Xn_{\sigma=\{xn\}}\} \prec W[X1, ..., Xn]$, $R_B(W, X \to A) \in G'^+$. That is for any restricted FD $g \in G$, we have $g \in G'^+$. So $G \subseteq G'^+$.□

As to the soundness and completeness of Algorithm EFFICIENT_PROPAGATE, we have the following results.

Theorem 5.2: Algorithm EFFICIENT_PROPAGATE is sound and complete to infer reasonable restricted FDs.

Proof: Given a transformation and a set of reasonable restricted FDs holding in the original relations of the transformation, Algorithm NAIVE_PROPAGATE and EFFICIENT_PROPAGATE compute the equivalent sets of restricted FDs. This follows from Lemma 5.4 and 5.6 directly. As Alg. NAIVE_PROPAGATE is sound and complete, so is Alg. EFFICIENT_PROPAGATE. □

5.4 Complexity Considerations of the EFFICIENT_PROPAGATE and CLOSURE Algorithms

As Alg. EFFICIENT_PROPAGATE does not compute restricted FD closures, given a set of restricted FDs G produced by Alg. EFFICIENT_PROPAGATE, we usually need (a similar algorithm to) Alg. CLOSURE (given in Section 4.3) to check whether a reasonable restricted FD follows from G. In this sub-section, we derive some results about the complexities of the EFFICIENT_PROPAGATE and CLOSURE algorithms resp. We first consider Alg. EFFICIENT_PROPAGATE, then CLOSURE.

In the worst case, we can design a transformation $T=<T_1, …, T_k>$ transforming a set of relations R_0 into R_k and a set of restricted FDs holding in R_0, such that the cardinality of a minimum cover of the restricted FDs holding in R_k is exponential to k. That is, the worst case time complexity of the EFFICIENT_PROPAGATE algorithm must be at least exponential to the number of transformation steps. Note the NAIVE_PROPAGATE algorithm is hyper-exponential as it computes restricted FD closure F^+ which takes time exponential to |F| for each transformation step.

In what follows, we first derive an upper bound for the algorithm and then present two classes of inputs for which EFFICIENT_PROPAGATE behaves polynomially.

Lemma 5.7: Given an input of Alg. EFFICIENT_PROPAGATE, let $T=<T_1, …, T_k>$ be a transformation, U be the set of attributes of the original relations of T, and F_0 be a set of restricted FDs holding in the original relations of T. If a is an upper bound for the cardinality of the set of restricted FDs G produced by a sub-algorithm during the execution of Alg. EFFICIENT_PROPAGATE, then the algorithm takes time $O(a^4|U|k)$.

Proof: During the i-th call of one of those 4 sub-algorithms to infer restricted FD for T_i, the worst case occurs while T_i is a set of *unfold* operators when less than a^3 sequences of restricted FDs *(f1, f2, f3)* are examined (see rule (10, 11) of Alg. INFER_UNFOLD). For each sequence (*f1, f2, f3*) the following operations are performed:

(a) checking whether *f1, f2* and *f3* produce a restricted FD according to rule (10, 11) of Alg. INFER_UNFOLD. This operation are feasible in O(|U|) time. Note although the set of attributes U would be changed during a transformation, the number of attributes in a restricted FD would always be O(|U|) during the execution of EFFICIENT_PROPAGATE. This will be followed in the rest of this section.

(b) Inserting the produced restricted FD into the set G. This requires less than O(|U|a) time.

The time to perform (a) and (b) for each *(f1, f2, f3)* local to the i-th call of one of those 4 sub-algorithms is thus O(a^4|U|). So the total time is O(a^4|U|k). □

A precise characterization of large and significant input-classes for which EFFICIENT_PROPAGATE runs in polynomial time is an interesting problem. In Theorem 5.3 and 5.4, we'll present two simple classes of "benign" inputs for EFFICIENT_PROPAGATE. The first class of inputs contains all possible inputs, for which the *unfold* operator sets in the given transformation is less a constant c. The second class of inputs is defined through constraints on the structure of F_0, the set of input restricted FDs holding in the original relations.

Theorem 5.3: Given an input of Alg. EFFICIENT_PROPAGATE, let T=<T_1, …, T_k> be a transformation, U be the set of attributes of the original relations of T, and F_0 be a set of restricted FDs holding in the original relations of T. If the number of *unfold* operator sets in T is less than a constant c, then the algorithm runs in polynomial time.

Proof: In general, there's at most one *split* operator in T, which generates a set of restricted FDs square in the size of input restricted FDs at most. Each of *unite* and *fold* operators generates a set of restricted FDs with the same size as that of input restricted FDs. Now let's consider *unfold*. Without loss of generality, suppose T_{i+1}, …, T_{i+k1} are *unfold* operators in T. Let F_i be the input restricted FDs in the original relations of T_{i+1}. We have $|F_{i+k1+1}|=O(|F_i|^{3^{k1}})$, which is polynomial in $|F_i|$ as we assume k1<c. Let a be the upper bound for the cardinality of the set of restricted FDs during the execution of Alg. EFFICIENT_PROPAGATE. Then a is polynomial in $|F_0|$. According to Lemma 5.7, the algorithm takes polynomial time w.r.t the number of input restricted FDs $|F_0|$, the number of attributes |U| and transformation steps k. □

Definition 5.5: Let F be a set of reasonable restricted FDs. The set **canonical(F)** of all canonical restricted FDs generated by F contains all reasonable restricted FDs f of the form $R(X \to A)$ from F^+ such that

(1) *Non-trivial.* Attribute A is not in X.

(2) *Left-reduced.* For no proper subset $Y \subset X$ it holds $R(Y \to A)$.

(3) For no super-set $R' \supset R$ it holds $R'(X \to A)$.□

Although $|F^+|$ is always exponential in |U|, and often exponential in |F|, |canonical(F)| can be very small and is polynomial in |U| or |F| in most cases of practical relevance. For instance, suppose F is a set of conventional FDs. If |canonical(F)| is exponential in |U| or |F| then there must exist an attribute of U which has exponentially many minimal keys. This is not a very common situation.

Theorem 5.4: Given an input of Alg. EFFICIENT_PROPAGATE, let T=<T_1, …, T_k> be a transformation, U be the set of attributes of the original relations of T, and F_0 be a set of restricted FDs holding in the original relations of T. If the size of canonical(F_0) is polynomial in $|F_0|$, then the algorithm runs in polynomial time.

Proof: For each restructuring operator set T_i in T, let F_{i+1} be the set of derived non-trivial restricted FDs in the transformed relations of T_i. If canonical($|F_0|$) is polynomial in $|F_0|$, we can easily prove that $|F_{i+1}| = O(|\text{canonical}(F_0)|)$ by induction. By applying Lemma 5.7, we know that Alg. EFFICIENT_PROPAGATE takes polynomial time in this case. □

As mentioned in the proof of Lemma 5.5, given a set of restricted FDs F, we can amend the CLOSURE algorithm to check whether a restricted FD follows from F. The time complexity of the amended algorithm is the same as that of CLOSURE. In what follows, we consider the cost to check whether a restricted FD follows from a set of restricted FDs produced by Alg. EFFICIENT_PROPAGATE.

Theorem 5.5: Given a transformation T=<T_1, …, T_k>, a set of original relations R_0 of T, a set of attributes U of R_0, and a set of restricted FDs F_0 for R_0 as inputs of Alg. EFFICIENT_PROPAGATE which satisfy the condition of Theorem 5.3 or 5.4, let G be the set of restricted FDs produced by Alg. EFFICIENT_PROPAGATE. If the number of *fold* operator sets in T is less than a constant c, then it takes polynomial time to check whether a reasonable restricted FD follows from G.

Proof: Let Z be the set of attributes occurred in some restriction attributes of the restricted FDs of G. Let m be the cardinality of Z, and d be an upper bound for the cardinalities of the domains of the attributes of Z. In the transformation T, only *unite* (Note there's at most one *unite* operator in T as T is a non-redundant transformation) and *fold* operators would change a reasonable restricted FD into a restricted FD with restriction attributes. So if the *fold* operator sets in T is less than a constant c, then m<c+1.

Furthermore, as the attributes of Z are computed by *fold* and *unite* transformations, the values of those attributes are attribute names or relation names of R_0. So d = max(|U|, $|R_0|$). |G| is polynomial in k, |U| and $|F_0|$ as the input of Alg. EFFICIENT_PROPAGATE satisfy the condition of Theorem 5.3 or 5.4. And as mentioned in the proof of Lemma 5.7, the change of the attribute set U during a transformation would not increase the complexity of algorithms. Consequently, according to Lemma 4.5, it takes polynomial time to check whether a reasonable restricted FD follows from G.□

6 Apply Restricted FDs to Works in Schematic Discrepancy

In Section 5, we have proposed algorithms to infer restricted FDs in schematic discrepant transformations. In this section, we'll study how to use those inferred restricted FDs. Specifically, we identify three scenarios to apply restricted FDs to works in schematic discrepancy; each sub-section is about one scenario.

6.1 Well-defined SchemaSQL View

In this sub-section, we'll see a problem of SchemaSQL views which may generate ambiguous results. We call those problematic views not "well-defined". In what follows, we first give a general definition of "well-defined SchemaSQL views" by extending the concept of information capacity [MIR93], then propose a method to automatically check whether a SchemaSQL view is well-defined by using FDs. We consider views for query purpose, not for update purpose.

Definition 6.1 ***(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* **à** *S2* is a many to one mapping, we call *V* is well-defined. □

Though a SQL view defines a mapping from instances of original relations onto 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. For example, in Example 2.3 (Sec 2.3), the number of relations in *DB3* makes no difference to the view definition. This is achieved by the use of relation-name variable *M* in the view statements. Generally, in SchemaSQL, for a query *Q* on a view $S \in S2$, we have:

$$Q(S) = Q(V(R)) = Q \circ V(R)$$

That is to say, the query *Q* against *S* is mapped to the unique query $Q \circ V$ against the original relation (or relation set) $R \in S1$ if *V* is a many to one mapping.

Example 6.1: Suppose in relation *Supply* (Figure 6.1), the FD *product, supplier, month* ® *price* holds. The SchemaSQL statements below define a view *SupView* (the database name is omitted for convenience):

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

As there's a variable *T.month* in the "*create view*" clause of the view definition, the view schema depends on the values of the *month* attribute in the *Supply* relation, i.e. *SupView(product, Jan, Feb)*. The instance of *SupView* can be computed according to SchemaSQL semantics (see Section 2.3 for an introduction). However, the produced *SupView* relation will have different instances (*SupView(I1)* or *SupView(I2)* in Figure 6.1) if we adopt different merge strategies in the interpretation of the semantics of the view. That is to say, the mapping from the original relations onto the view relations is many to many, so the view is not well-defined. □

Supply

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

SupView(I1)

product	Jan	Feb
p1	100	105
p1	95	97

SupView(I2)

product	Jan	Feb
p1	100	97
p1	95	105

Figure 6.1: Ambiguous SchemaSQL view: *SupView* may have one of two instances I1 and I2

The following theorem gives a 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 single relations without aggregations. The result can be extended to general SchemaSQL views readily.

Theorem 6.1: A SchemaSQL view is well-defined if it satisfies the following condition: if the output schema declaration through the *create view* statement of the view definition has a form "*DB::R*(A_1, ..., A_n, *B*)", where *DB::R* is the database and relation name of the view, A_1, ..., A_n are attribute names, and *B* is a variable ranging over a set of values {b_1, ..., b_m}, then the following FDs hold in *DB::R*.

$$A_1, \ldots, A_n \rightarrow b_i, \quad i = 1, \ldots, m$$

Proof: As mentioned in Section 2.3, the semantics of a SchemSQL view could be interpreted in 4 steps. The first 3 steps can be done in a deterministic way, i.e., each of the 3 steps is a many to one mapping, while the last step of merging is not in general. Theorem 6.1 requires the FDs $A_1, \ldots, A_n \rightarrow b_i$, $i = 1, \ldots, m$, hold in the view relation. That is, given the values of $A_1, \ldots, A_n$, we have at most one value for each attribute b_i. So in the last step of merging, the set of tuples with the same values on $A_1, \ldots, A_n$ in the allocated table are merged into one tuple in the view relation. Now the merging step is deterministic, and the target relation is unique. □

Note according to SchemaSQL syntax [LSS01], there's at most one variable in the attribute list of the output schema declaration through a *create view* statement. And Theorem 6.1 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 6.1 could be used to check all the SchemaSQL views which generate single relations without aggregations.

According to Theorem 6.1, in order to check whether a SchemaSQL view is well-defined, we need to infer restricted FDs holding in the view relation. SchemaSQL queries/views can be implemented by use of restructuring operators and relational algebra [LSS99]. Consequently, we should develop algorithms to infer restricted FDs in application of relational algebra (selection, projection, join, and so on) besides restructuring operators, which are suppressed here. We hereby give an example to describe this process.

Example 6.2: The view of Example 6.1 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 view *SupView*. As step (1) project out the *supplier* attribute, the given FD *product, supplier, month*→*price* is lost after the projection. Consequently, no restricted FD holds in *SupView*, which means the view is not well-defined.

On the other hand, if the view schema contains attribute *supplier*, i.e., *SupView(product, supplier, Jan, Feb)*, then the view is implemented by performing *unfold(Supply, month, price)*. Using Algorithm INFER_UNFOLD (rule (2)), we have *product, supplier*→*Jan, Feb* in *SupView*. Now the view is well-defined. □

6.2 Normalizing Transformed Relations

In schema transformation, people could suppose original schemas of component databases are in good normal form (BCNF or 3NF) if they were well designed. But how are the transformed (integrated) schemas? We find a schematic discrepant transformation may introduce redundancy in the transformed relations. Redundancy not only wastes storage space, but also causes update, insertion and deletion anomalies, and therefore should be removed. People have developed decomposition and synthesizing methods to convert a relation to BCNF or 3NF given FDs holding in that relation [RG99].

Given a transformation, using Algorithm EFFICIENT_PROPAGATE, we can infer the set of restricted FDs, say F_k, in the transformed relations R_k from a set of restricted FDs, say F_0, in the original relations R_0. To normalize the relations of R_k, we need to extract FDs from F_k. In general, it takes time exponential in the size of F_k to extract all the FDs from F_k. Actually, let G be a minimum cover of the FDs implied by F_k. The size of G could be exponential in the size of F_k. However, if each restricted FD in F_0 is a FD holding either in the union of all the relations of R_0 or in each relation of R_0, then the problem can be solved in polynomial time. The general algorithm is suppressed here. We hereby give an example to describe the process.

Example 6.3: Suppose we want to integrate two relations keeping book information of two bookstores, i.e., *BS1(isbn, title, price)* and *BS2(isbn, title, price)*. Suppose *store* is an attribute name and *dom(store)* = {*BS1*, *BS2*}. In each relation, *isbn* is the key. Suppose the following restricted FDs hold: {*BS1,BS2*}(*isbn*→*title*) and *BSi*(*isbn*→*price*) for each *i = 1, 2*. That is to say, a book's title is only dependent on the isbn number, while price is also dependent on bookstores.

Although the two relations have the same schema, integrating them into a relation with the same schema (as many existing schema integration approaches do) cause value inconsistency on the *price* attribute. A better solution is to integrate them into *Book(store, isbn, title, price)* using *unite({BS1, BS2}, store)*. We can infer restricted FDs in the integrated *Book* relation using Algorithm INFER_UNITE: *Book(isbn ® title)* and *Book(store*$_{\sigma=\{BSi\}}$*, isbn ® price)* for each *i = 1, 2*, which imply FDs *isbn ® title* and *store, isbn ® price* in the *Book* relation, according to Axiom 4.1 (rule (6) and (11)).

According to the inferred FDs, we know that the integrated relation is not in 2nd normal form. Consequently, for any book sold in both stores, the *title* of the book is repeated twice in the integrated relation. We can decompose the *Book* relation into two relations: *Book_price(store, isbn, price)* and *Book_title(isbn, title)*. □

6.3 Formulating Global Constraints in Schema Integration

Generally, in schema integration, global constraints holding in integrated schemas can be used to verify the integrity of data from different sources, to optimize queries on the integrated schemas, or to validate update transactions at the global level [RPG95]. Our method (Algorithm EFFICIENT_PROPAGATE) provides a way to formulate restricted FDs for integrated schemas in the presence of schematic discrepancy.

7 Related Works

Relational dependencies are studied much in 1970s and early 1980s. Motivated by practical examples, people have introduced *FDs, multivalued dependencies, embedded multivalued dependencies, inclusion dependencies*, etc. To generalize those dependencies and study two of the fundamental issues in dependency theory, deciding logical implication and constructing axiomatization, some unifying frameworks are proposed, presented in the paradigms of restricted forms of first-order logic, tableaux or algebraic terms. An interesting and powerful method is to use tableaux (a table form representation) to present constraints, and using "chase" (a procedure based on the successive application of constraints to tableaux) to analyze implication [AHV95]. Axiomatizations for important subclasses have been developed, again using the chase. However, the existing tableaux and chase paradigm does not subsume our proposal of restricted FDs which have restrictions on attribute values and hold over the union of a set of relations. And the inference rules developed in the tableaux paradigm cannot be used to infer restricted FDs.

Another kind of extension to FDs in 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* [Ling01], *afunctional dependencies* [BP83] and *partial FDs* [BCCM02]. 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 defined in single relations, while qualified FDs may hold over the union of a set of relations.

The issue of inferring view dependencies is introduced in [AHV95, Got87]. However, their works are based on views defined using relational algebra. And the dependencies they considered does not subsume qualified FDs. Recently, some works [LL97, RPG95, VA96] have been done on the derivation of constraints on the integrated schema from constraints on the component schemas during 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.

8 Conclusion

There're 3 contributions in this paper: (1) We proposed restricted FDs to formalize FD-like ICs in schematic discrepant databases. We gave a sound and complete set of inference rules for restricted FDs. (2) We studied the propagation of restricted FDs in schematic discrepant transformations. Our algorithms are complete in the sense that they compute all the restricted FDs holding in transformed relations from restricted FDs in original relations. Although the problem is at least exponential inherently, our algorithm behaves polynomially for two classes of inputs. (3) We showed 3 applications of restricted FDs in the context of multi-database interoperability: (a) In SchemaSQL, we introduced "well-defined SchemaSQL views", and gave a method to verify it by using FDs. (b) A schematic discrepant transformation could introduce redundancy in the transformed relations, which leads to update, insertion and deletion anomalies. We gave a method to infer FDs in the transformed relations which can be used to normalize those redundant relations. (c) We can use restricted FDs to verify the integrity of data, optimize queries, and validate updates on integrated schemas.

Reference

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases, pp173-187, 216-235. *Addison-Wesley*, 1995.

[Alb00] J. Albert. Theoretical Foundations of Schema Restructuring in Heterogeneous Multidatabase Systems. *CIKM*, 2000.

[ASX01] R. Agrawal, A. Somani, and Y. R. Xu. Storing and querying of e-commerce data. *J. VLDB*, 2001.

[BCCM02] F. Berzal, J. C. Cubero, F. Cuenca, J. M. Medina. Relational decomposition through partial functional dependencies. *DKE*, 2002.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for schema integration. *CN computing surveys*, 1986.

[BP83] P. De Bra and J. Paredaens. Conditional dependencies for horizontal decompositions. *ICALP*, 1983.

[GLS96] M. Gyssens, L. Lakshmanan, and S. N. Subramanian. Tables as a paradigm for querying and restructuring. *PODS,* 1996.

[Got87] G. Gottlob. Computing covers for embedded functional dependencies. *SIGMOD,* 1987.

[HK00] C. N. Hsu and C. A. Knoblock. Semantic query optimization for query plans of heterogeneous multidatabase systems. *TKDE*, 2000.

[KLK91] R. Krishnamurthy, W Litwen, and W. Kent. Language features for interoperability of databases with schematic discrepancies. *SIGMOD*, 1991.

[Ling01] T. W. Ling. Extending classical functional dependencies for physical database design (lecture notes). http://www.comp.nus.edu.sg/~lingtw/cs4221/extended.fds.pdf

[LL95] M. L. Lee and T. W. Ling. Resolving Structural Conflicts in the Integration of Entity Relationship Schemas. *OOER conf.*, 1995.

[LL96] T. W. Ling, M. L. Lee. Issues in an Entity-relationship based federated database system. *CODAS*, 1996.

[LL97] M. L. Lee and T. W. Ling. Resolving constraint conflicts in the integration of ER schemas. *ER conf.*, 1997.

[LSS99] L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. On efficiently implementing schemaSQL on SQL database system. *VLDB conf.*, 1999.

[LSS01] L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. SchemaSQL—an extension to SQL for multidatabase interoperability. *TODS*, 2001.

[Nic78] J. M. Nicolas. First order logic formalization for functional, multivalued, and mutual dependencies. *SIGMOD*, 1978.

[Mil98] R. J. Miller. Using schematically heterogeneous structures. *SIGMOD*, 1998.

[MIR93] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. *VLDB conf.*, 1993.

[RG99] R. Ramakrishnan and J. Gehrke. Database Management Systems, 2nd Ed, pp427, 438-444. *McGraw-Hill*, 1999.

[RPG95] M. P. Reddy, B. E. Prasad, and A. Gupta. Formulating global integrity constraints during derivation of global schema. *Data & Knowledge Engineering*, 1995.

[SG90] A. P. Sheth and S. K. Gala, Federated database systems for managing distributed, heterogenous, and autonomous databases. *ACM Computing Surveys*, 1990.

[VA96] M. W. W. Vermeer and P. M. G. Apers. The role of integrity constraints in database interoperation. *VLDB conf.*, 1996.

Appendix

Algorithm INFER_UNITE: *Inference of restricted FDs for unite operator.*

INPUT: Let $b_i(A_1, ..., A_n)$ $(i = 1, ..., m)$ be original relations of *DB1*, and $R(A_1, ..., A_n, B)$ be the transformed relations of *DB2* using $unite(\{b_1, ..., b_m\}, B)$.

Let F be a set of restricted FDs holding in the relations $\{b_1, ..., b_m\}$ of *DB1*.

OUTPUT: a set of restricted FDs, G, holding in *DB2::R*.

METHOD: Let $\overline{X}$ be a mixed set of regular and restriction attributes from $\{A_1, ..., A_n\}$, and $A \in \{A_1, ..., A_n\}$. We compute the restricted 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 $b_i(\overline{X} \rightarrow A) \in F$ for some $b_i \in dom(B)$, then $R(B_{\sigma=\{b_i\}}, \overline{X} \rightarrow A) \in G$. □

Algorithm INFER_UNFOLD: *Inference of restricted FDs for a set of unfold operators.*

INPUT: Let $R_i(A_1, ..., A_n, B, C)$ $(i=1,...,l)$ be original relations of *DB1* with FD $A_1, ..., A_n, B \rightarrow C$ holding in each relation $DB1::R_i$, and $R_i(A_1, ..., A_n, b_1, ..., b_m)$ be the transformed relation of *DB2* using $unfold(DB1::R_i, B, C)$ for each $i=1, ..., l$.

Let F be a set of restricted FDs holding in the relations $\{R_1, ..., R_l\}$ of *DB1*.

OUTPUT: a set of restricted FDs, G, holding in the relations $\{R_1, ..., R_l\}$ of *DB2*.

METHOD: Let $\overline{X}$, $\overline{Y}$ and $\overline{Z}$ be mixed sets of regular and restriction attributes from $\{A_1, ..., A_n\}$, and $A \in \{A_1, ..., A_n\}$. Let $R^{(1)}$ ($R^{(2)}$) be a subset of relations of $\{R_1, ..., R_l\}$ from *DB1* (*DB2*), and $R^{(1)}$ and $R^{(2)}$ contain the same set of relation names. We compute the restricted FDs in G using the following rules:

(1) If $R^{(1)}(B_{\sigma=\{b_i\}}, \overline{X} \rightarrow C) \in F$, then $R^{(2)}(\overline{X} \rightarrow b_i) \in G$.

(2) If $R^{(1)}(B, \overline{X} \rightarrow C) \in F$ or $R^{(1)}(\overline{X} \rightarrow C) \in F$, then $R^{(2)}(\overline{X} \rightarrow b_i) \in G$ for each $b_i \in dom(B)$.

(3) If $R^{(1)}(B_{\sigma=\{b_i\}}, \overline{X}, C \rightarrow A) \in F$, then $R^{(2)}(\overline{X}, b_i \rightarrow A) \in G$.

(4) If $R^{(1)}(B, \overline{X}, C \rightarrow A) \in F$ or $R^{(1)}(\overline{X}, C \rightarrow A) \in F$, then $R^{(2)}(\overline{X}, b_i \rightarrow A) \in G$ for each $b_i \in dom(B)$.

(5) If $R^{(1)}(B_{\sigma=\{b_i\}}, \overline{X} \rightarrow A) \in F$, then $R^{(2)}(b_{i\ \sigma=dom(b_i)}, \overline{X} \rightarrow A) \in G$.

(6) If $R^{(1)}(B, \overline{X} \rightarrow A) \in F$, then $R^{(2)}(b_{i\ \sigma=dom(b_i)}, \overline{X} \rightarrow A) \in G$ for each $b_i \in dom(B)$.

(7) If $R^{(1)}(\overline{X} \rightarrow A) \in F$, then $R^{(2)}(\overline{X} \rightarrow A) \in G$.

(8) If $R^{(1)}(\overline{X} \rightarrow B) \in F$ and $R^{(1)}(\overline{Y}, B \rightarrow A) \in F$, then $R^{(2)}(\overline{X}, \overline{Y} \rightarrow A) \in G$.

(9) If $R^{(1)}(\overline{X} \rightarrow C) \in F$ and $R^{(1)}(\overline{Y}, C \rightarrow A) \in F$, then $R^{(2)}(\overline{X}, \overline{Y} \rightarrow A) \in G$.

(10) If $R^{(1)}(\overline{X} \rightarrow B) \in F$, $R^{(1)}(\overline{Y}, B \rightarrow C) \in F$ and $R^{(1)}(\overline{Z}, C \rightarrow A) \in F$, then $R^{(2)}(\overline{X}, \overline{Y}, \overline{Z} \rightarrow A) \in G$.

(11) If $R^{(1)}(\overline{X} \rightarrow C) \in F$, $R^{(1)}(\overline{Y}, C \rightarrow B) \in F$ and $R^{(1)}(\overline{Z}, B \rightarrow A) \in F$, then $R^{(2)}(\overline{X}, \overline{Y}, \overline{Z} \rightarrow A) \in G$.

(12) If $R^{(1)}(\overline{X} \rightarrow C) \in F$, then $R^{(2)}(\overline{X1}, b_i \rightarrow b_j) \in G$ for any $b_i, b_j \in dom(B)$. $\overline{X1}$ is the set of all the restriction attributes in $\overline{X}$. □

Algorithm INFER_FOLD: *Inference of restricted FDs for a set of fold operators.*

INPUT: Let $R_i(A_1, ..., A_n, b_1, ..., b_m)$ $(i=1,...,l)$ be original relations of *DB1* with FD $A_1, ..., A_n \rightarrow b_1, ..., b_m$ holding in each relation $DB1::R_i$, and $R_i(A_1, ..., A_n, B, C)$ be the transformed relation of *DB2* using $fold(DB1::R_i, B, C)$ for each $i=1, ..., l$.

Let F be the set of restricted FDs holding in the relations $\{R_1, ..., R_l\}$ of *DB1*.

OUTPUT: a set of restricted FDs, G, holding in the relations $\{R_1, ..., R_l\}$ of *DB2*.

METHOD: Let $\overline{X}$ be a mixed set of regular and restriction attributes from $\{A_1, ..., A_n\}$, and $A \in \{A_1, ..., A_n\}$. Let $R^{(1)}$ ($R^{(2)}$) be a subset of relations of $\{R_1, ..., R_l\}$ from *DB1* (*DB2*), and $R^{(1)}$ and $R^{(2)}$ contain the same set of relation names. We compute the restricted FDs in G according to the following rules:

(1) If $R^{(1)}(\overline{X} \rightarrow b_i) \in F$ then $R^{(2)}(B_{\sigma=\{b_i\}}, \overline{X} \rightarrow C) \in G$.

(2) If $R^{(1)}(\overline{X}, b_i \rightarrow A) \in F$ then $R^{(2)}(B_{\sigma=\{b_i\}}, \overline{X}, C \rightarrow A) \in G$.

(3) If $R^{(1)}(b_{i\ \sigma=dom(b_i)}, \overline{X} \rightarrow A) \in F$ then $R^{(2)}(B_{\sigma=\{b_i\}}, \overline{X} \rightarrow A) \in G$.

(4) If $R^{(1)}(\overline{X} \rightarrow A) \in F$ then $R^{(2)}(\overline{X} \rightarrow A) \in G$. □