数理解析研究所講究録
第436巻 1981年 1-26

Dependency Integration of Locally Independent Relational Databases into a Distributed Database

Katsumi Tanaka
College of Liberal Arts, Kobe University
Nada, Kobe 657, Japan
Yahiko Kambayashi
Dept. of Information Science, Kyoto University
Sakyo, Kyoto 606, Japan

INTRODUCTION

In the research on distributed database systems, much effort has been devoted to the study of systems architecture, query processing, concurrency control, multiple copy update problems and file allocation etc. These results have been also used for the actual implementations of several distributed database management systems (for short, D-DBMS's). Basically, D-DBMS's can be classified into

(1) homogeneous type (for example, SDD-1[1])
(2) heterogeneous type (for example, POLYPHEME[2], Multibase[16])

As for the logical distributed database design, the following approaches are known:

(i) top-down design approach
(ii) bottom-up design approach

The top-down design approach often suits well for a homogeneous type D-DBMS. The bottom-up design approach is often adopted in heterogeneous type D-DBMS's since most of heterogeneous type D-DBMS's are constructed in order to integrate several locally independent DBMS's and databases which already exist.

In this paper, we mainly discuss a logical integration problem of local relational database constraints. We assume (1) a homogeneous type D-DBMS (a relational DBMS) and (ii) the bottom-up design approach in this paper. Our choice of the combination

(1)-(ii) is motivated from the following reasons:
(a) When we choose a heterogeneous type D-DBMS and the bottom-up design approach (that is, (2)-(ii)), it is also necessary to consider how to integrate homogeneous local databases by the bottom-up design approach since heterogeneous local databases are often logically translated into homogeneous local databases[2,3,16]. Thus, our results in this paper are also useful for the construction of heterogeneous distributed database systems.
(b) As described later, the bottom-up design approach often offers more flexibility than the top-down approach in maintaining constraints of local databases, and in this sense, this approach is more practical.
(c) It is possible to transform CODASYL-type databases or hierarchical model databases into relational databases[3]. Furthermore, several commercial relational DBMS's have become available on many kinds of computers.

Each local relational database has its own semantics and constraints, and therefore, the logical integration of these local databases must be performed very carefully. As the result of the logical integration, one or more global views are provided to users. In this case, it is important to consider what kinds of semantic constraints are provided to users by the global views of the distributed database. Some semantic constraints imposed on a local database may be violated when creating a global view. Furthermore, some semantic constraints, which are not imposed on any local database, may appear in a global view. When some semantic constraint on a local database is violated and does not hold on a global view, the distributed database designer must decide whether or not such a violation is allowable (or meaningful), and he must find a method for solving the problem if it is not allowed. Furthermore, semantic constraints imposed on global views are useful information for performing distributed query processing efficiently.

In this paper, we discuss a method for calculating semantic constraints (especially, dependency constraints) imposed on a global view. As shown later, some join dependency holding on a local

relational database may be violated when creating a global view by intgrating local databases by join, projection and renaming operations. We show the relationship between a class of semantic constraints of a global view and a class of semantic constraints of local relational databases. A method is given for testing whether or not an embedded join dependency[4] holds on a global view. This method is a simple modification of the 'chase' procedure by Maier et al[4]. Immediately from our method, we also show a necessary and sufficient condition for a join dependency holding on a local database to hold on a global view.

BASIC CONCEPTS

A relation is defined as a finite set of mappings from the set of attributes to the corresponding domains. Attributes are symbols taken from a given finite set of symbols. A domain is simply a set of all the possible values, which can appear as the corresponding attribute values. We use A,B,C,... (possibly with subscripts) to denote single attributes, and ...,X,Y,Z (possibly with subscripts) to denote sets of attributes. For a set of attributes $R=\{A_1,...,A_n\}$ and an associated domain D_i for each attribute A_i, a relation r on R is a finite set of mappings t such that $t: \{A_1,...,A_n\} \rightarrow D$, where D is the union of the D_i's. The mapping must map each attribute in R to a member of its corresponding domain. Hereafter, we often represent a set of attributes by omitting commas and set braces, so that, for example, $\{A_1,...,A_n\}$ is written as $A_1...A_n$. The union of sets of attributes is also represented by concatenation, e.g., $X \cup Y \cup Z$ is written as XYZ.

Let R be a given finite set of attributes. A local (database) scheme (for short, LS) at site i (i=1...n) is a subset of R of attributes. A global database scheme for LS's $R_1,...,R_n$ is defined as a relational algebra expression using restriction, join, projection etc.[5] for the local schemes. A

global join scheme (for short, GJS) for LS's is defined by only join operators as follows:

$$R_1*\ldots*R_n \text{ (or denoted by } \overset{n}{\underset{i=1}{*}}R_i).$$

In this paper, we consider the semantic constraints of only GJS's.

For a mapping t on a set R of attributes and a set $X \subseteq R$, we denote the restriction of t to X by t[X]. Let r be a relation on R. The projection of r on X, denoted by r[X], is the set $r[X]=\{t' \mid$ for some $t \in r$, $t'=t[X]\}$. The projection r[X] is a relation on X whose elements are the restrictions of the mappings in r to X. For $1 \le i \le n$, let r_i be a relation on the set R_i of attributes. The (natural) join of $r_1,\ldots,r_n$, denoted by $r_1*\ldots*r_n$ or $\overset{n}{\underset{i=1}{*}}r_i$, is the relation on $\overset{n}{\underset{i=1}{\cup}}R_i$ defined by $\overset{n}{\underset{i=1}{*}}r_i=\{t \mid$ t is a mapping on $\overset{n}{\underset{i=1}{\cup}}R_i$ such that $t[R_i]$ is in r_i for all i, $1 \le i \le n\}$.

An semantic constraint c for a set R_i of attributes is a predicate, that assigns to each relation on R_i either 'true' or 'false'. When a relation r_i is assigned to 'true', r_i is said to satisfy c, and c is said to hold in r_i. For a given set C_i of semantic constraints, we denote the set of relations on R_i that satisfy every constraint in C_i by $SAT(R_i,C_i)$. We say a set C_i of semantic constraints holds in R_i if and only if every meaningful relation on R_i must satisfy C_i.

A functional dependency (FD)[6] is a semantic constraint denoted by $X \rightarrow Y$. It can be defined for every relation on R such that $XY \subseteq R$. The FD $X \rightarrow Y$ holds in relation r if and only if, for all mappings t_1, t_2 in r, $t_1[X]=t_2[X]$ implies $t_1[Y]=t_2[Y]$. An embedded join dependency (EJD)[4,7] is a semantic constraint denoted by $*[S_1,\ldots,S_m]$, which can be defined for every relation on R such that $R \supseteq \overset{m}{\underset{i=1}{\cup}}S_i$. The EJD $*[S_1,\ldots,S_m]$ holds in relation r if and only if

$$\overset{m}{\underset{i=1}{*}}r[S_i]=r[\overset{m}{\underset{i=1}{\cup}}S_i].$$

If the union of S_i's is equal to R, then $*[S_1,\ldots,S_m]$ is said to be a join dependency (JD) on R. Let d be a single dependency that can

be defined on R. The dependency d is said to be logically implied by a set D of dependencies on R if and only if d holds in any relation on R that satisfies all the dependencies in D. If D logically implies d, then we denote it by D |= d.

Let $R_1*\ldots*R_n$ be a GJS such that $\bigcup_{i=1}^{n} R_i=R$ and D_i denotes a set of FDs and EJDs that hold in R_i for each i, $1\leq i\leq n$.

(1) Let $X \rightarrow Y$ be an arbitrary FD such that $XY \subseteq R$. We say $R_1*\ldots*R_n$ preserves $X \rightarrow Y$ if and only if $*_{i=1}^{n} r_i$ satisfies $X \rightarrow Y$ for any distributed database $\{r_1,\ldots,r_n\}$ such that r_i belongs to $SAT(R_i,D_i)$ for each i.

(2) Let $*[S_1,\ldots,S_m]$ be an arbitrary EJD such that $R \supseteq \bigcup_{j=1}^{m} S_j$. We say $R_1*\ldots*R_n$ preserves $*[S_1,\ldots,S_m]$ if and only if $*_{i=1}^{n} r_i$ satisfies $*[S_1,\ldots,S_m]$ for any distributed database $\{r_1,\ldots,r_n\}$ such that r_i belongs to $SAT(R_i,D_i)$ for each i.

Here, $SAT(R_i,D_i)$ denotes a set of all the relations on R_i that satisfy every dependency in D_i.

EXAMPLE

Let us consider a distributed database consisting of two sites. Site 1 has an LS R_1={DEPT, RES, COM, DATE} and site 2 has an LS R_2={RES, COM, SOFT, LAN}. DEPT denotes a departmant name, RES a researcher name, COM a computer, DATE an installed date, SOFT a software name, LAN a programming language to write the corresponding software. Fig.1 shows example relations r_1 on R_1 and r_2 on R_2. Every tuple (d,r,c,i) in r_1 means that a researcher r belongs to department d and that computer c was installed at department d on date i. A researcher may belong to more than one department. A department may have more than one researcher. More than one department may have the same type of computer. Each installed date of a computer is uniquely determined by the department name and the computer name. Then, we have the following dependency constraints for R_1:

FD: {DEPT,COM} → DATE,

JD: *[{DEPT,RES}, {DEPT,COM,DATE}].

Each tuple (r,c,s,p) in r_2 means that researcher r developed software s on computer c by programming language p. We assume that the following FD holds in R_2:

FD: {RES,COM,SOFT} → LAN.

The relation r_1 satisfies {DEPT,COM} → DATE and *[{DEPT,RES}, {DEPT,COM,DATE}]. The relation r_2 satisfies {RES,COM,SOFT} → LAN.

If a distributed database designer defines R_1*R_2 as a GJS, then the relation r_1*r_2 in Fig.2 becomes a current instance for the global view. It is easily proved that the relation r_1*r_2 does not satisfy the EJD *[{DEPT,RES}, {DEPT,COM.DATE}], while r_1*r_2 satisfies the FDs {DEPT,COM} → DATE and {RES,COM,SOFT} → LAN. In this sense, the EJD *[{DEPT,RES}, {DEPT,COM,DATE}] cannot be preserved by the GJS R_1*R_2. Since it is easily proved that any FD holding in an LS is preserved by any GJS containing the LS, the two FDs above are preserved by R_1*R_2.

If the distributed database designer wishes to have a global view, in which any tuple (d,r,c,i,s,p) means that researcher r developed a software s by language p for computer c which was installed on date i at his department d, this GJS agrees with his intention.

If he wishes to preserve the EJD *[{DEPT,RES}, {DEPT,COM,DATE}] on a global view, one of the following renamings of attributes can be used:

(1) Rename attribute RES of R_2 into DEV(eloper) and create the GJS R_1*R_2.

(2) Rename attribute COM of R_2 into MAC(hine for the software) and create the GJS R_1*R_2.

(3) Perform the renamings in (1) and (2), and create the GJS R_1*R_2. (Renamings of attributes in R_1 are also possible).

Fig.3 shows a relation r_1*r_2 in which r_2 is a relation on R_2={DEV,COM,SOFT,LAN}. In this relation, we can find that the EJD *[{DEPT,RES}, {DEPT,COM,DATE}] is satisfied. Let $r=r_1*r_2$. Then, the

join r[{DEPT,RES}]*r[{DEPT,COM,DATE}] is equal to r[{DEPT,RES,COM,DATE}]. Therefore, r satisfies the EJD *[{DEPT,RES}, {DEPT,COM,DATE}]. r also satisfies the FDs shown above.

The semantic constraint *[{DEPT,RES}, {DEPT,COM,DATE}] is important at Site 1 since it means that
(a) there is a many-to-many correspondence between DEPT and RES,
(b) there is a many-to-many correspondence between DEPT and {COM,DATE},
(c) RES and {COM,DATE} are mutually independent.
Therefore, this semantic constraint is maintained at Site 1 for several update commands. As shown previously, however, if a distributed database designer regards that the constraint is unnecessary in his global scheme, the semantic constraint is violated as shown in Fig.2. In this sense, each local database has its own semantics and is semantically closed, and the semantics can be maintained independently from what global schemes are defined from the local databases. This offers more flexibility for the maintenance of local databases.

It is also important in distributed query processing to calculate semantic constraints on global views. For example, the JD *[XY,XZ], which holds in a global view or in a user's query expression, implies that the resultant data can be sent to the user in a more reduced form. That is, the data (x_1,y_1,z_1), (x_1,y_1,z_2), (x_1,y_2,z_1), (x_1,y_2,z_2) can be transferred as $x_1/y_1,y_2/z_1,z_2/$ if we know that *[XY,XZ] holds in a result. Furthermore, when a user issues a query, if a D-DBMS can calculate semantic constraints preserved on the user's query expression, the user can understand the semantic of his query well.

As shown in this example, a GJS must be created very carefully since the the semantics of a GJS depends very much on what dependencies can be preserved by the GJS. Therefore, it is

important to test whether or not a dependency is preserved by a defined GJS, and to solve the problem of the violation on a dependency if it is not allowed.

TESTING CONSTRAINTS ON A GJS

The example of the previous section shows that the following issues are important in the logical integration of local relational databases:

(1) To specify correctly a set of dependency constraints which must be preserved by a GJS.

(2) To test whether or not a specified dependency can be preserved by the GJS.

(3) To rename some attributes in some LS's so that every specified dependency may be preserved by the GJS.

As for (1), it is important to consider what class of dependency constraints can be preserved by a GJS for given LS's. First, we will show some conditions, which clarify the relationship between the class of preserved dependencies and the local dependency constraints. As for (2), a testing algorithm is required, which actually tests whether or not a specified dependency is preserved. We will show a method for testing whether or not an EJD is preserved by modifying the 'chase' procedure[4]. As for (3), it is necessary to find a condition for a specified, but not preserved dependency to be preserved after renaming some attributes. Such a condition can be obtained immediately from our testing method.

Let $\{R_1,\ldots,R_n\}$ be an arbitrary set of LS's, where D_i denotes a set of FDs and EJDs that hold on R_i $(i=1,\ldots,n)$. The following theorem provides a necessary condition for an FD or an EJD to be preserved by the GJS $R_1*\ldots*R_n$.

Theorem 1[15]

If $\overset{n}{\underset{i=1}{*}}R_i$ preserves a dependency d (FD or EJD), then d must be

logically implied by the union of $\{*[R_1,\ldots,R_n]\}$ and D_i's.

Theorem 1 states that the set of all the dependencies preserved by the GJS must be contained in the closure of $\bigcup_{i=1}^{n} D_i \cup \{*[R_1,\ldots,R_n]\}$. Here, the closure of a given set of dependencies means a set of all the FDs and EJDs that are logically implied by the given set. It should be noted that the converse of Theorem 1 does not always hold since some JD in some D_i may be violated, as shown in the previous section.

The following theorem provides a sufficient condition for an FD or an EJD to be preserved by the GJS $R_1*\ldots*R_n$.

Theorem 2 [15]

Let F_i denote a set of all the FDs that are logically implied by D_i. For a given dependency d (FD or EJD), if the union of F_i's (i=1,...,n) and $\{*[R_1,\ldots,R_n]\}$ logically implies d, then $*_{i=1}^{n} R_i$ preserves d.

For a given GJS $*_{i=1}^{n} R_i$, let SPECIFIED($*_{i=1}^{n} R_i$) denote a set of FDs and EJDs that a database designer wishes to preserve by the GJS. Let PRESERVED($*_{i=1}^{n} R_i$) denote a set of all the preserved dependencies (FDs and EJDs) by the GJS. Generally, the relationships among SPECIFIED($*_{i=1}^{n} R_i$), PRESERVED($*_{i=1}^{n} R_i$), D_i's and F_i's can be illustrated as shown in Fig.4.

As a special case, if each D_i of dependency constraints is equivalent to a certain set of FDs, we can prove that the condition of Theorem 1 becomes a necessary and sufficient condition as follows:

Corollary 1 [15]

Assume that each D_i is equivalent to a certain set F_i of FDs. That is, D_i is logically implied by F_i and F_i is also logically implied

by D_i. In this case, a necessary and sufficient condition for a dependency d (FD or EJD) to be preserved by GJS $\overset{n}{*}_{i=1} R_i$ is that the union of $\{*[R_1,\ldots,R_n]\}$ and D_i's logically implies d.

The result in Corollary 1 is strongly related to the result of Beeri and Rissanen[7,10]. Beeri and Rissanen has already shown the similar result under the (pure) universal relation assumption[11]. That is, each (distributed) database $\{r_1,\ldots,r_n\}$ must be a set of projections of a universal relation. The major difference of our work from theirs is that we do not have such an assumption. An example to illustrate the difference will be shown later in Example 1.

Before describing our testing method, we briefly summarize some preliminary concepts such as 'tableau' and 'chase' since our method is based on a modified chase method. The definitions of tableau and chase follow the papers by Maier et al.[4] and by Sciore[8].

A tableau for a set R of attributes is a set of rows, each with one variable for each attribute in R. The variables are either distinguished or nondistinguished. In any tableau, each variable can be associated with only one attribute, and there can only be one distinguished variable associated with each column. We use a_i for distinguished variables, and b_i for nondistinguished variables. Let T be a tableau, K the set of variables, and D the union of the domains of R. A valuation ρ of T is a mapping from K to D such that $\rho(k) \in D_i$ whenever k is associated with attribute A_i and the domain of A_i is D_i. Valuations are extended to rows in the following way: $\rho(k_1,\ldots,k_p)=\rho(k_1)\ldots\rho(k_p)$. The chase process is performed for a tableau T by the following FD-rule and JD-rule:

FD-rule For a given FD $X \rightarrow A$, whenever T has rows w_1 and w_2 that agree in all X-columns but does not agree in A-column, replace one of the A-column variable in T by the other A-column variable, and remove duplicate rows. If one of the A-column variables is distinguished, the other one must be renamed to that distinguished variable. If both are nondistinguished, rename the variable with larger subscript to be the variable with smaller subscript.

JD-rule For a given JD $*[Y_1,\ldots,Y_m]$ for T, add a row w to T if T contains rows $w_1,\ldots,w_m$ (not necessarily distinct) such that for all $1\le j\le m$, w_j agrees with w on Y_j.

[Testing Algorithm for Preserving EJDs][9,15]

Inputs: (1) Local schemes (LS's) $R_1,\ldots,R_n$, (2) sets of dependencies (FDs and JDs) $D_1,\ldots,D_n$ such that D_i holds in R_i for each i, $1\le i\le n$, and (3) an EJD $*[S_1,\ldots,S_m]$ such that $S=\bigcup_{j=1}^{m} S_j$, $R=\bigcup_{i=1}^{n} R_i$ and $S \subseteq R$.

Output: If $\overset{n}{*}_{i=1} R_i$ preserves $*[S_1,\ldots,S_m]$, then 'YES' otherwise 'NO'.

Step 1: (Construction of an initial tableau T) Construct an initial tableau T for $*[S_1,\ldots,S_m]$ such that (a) T consists of m rows $w_1,\ldots,w_m$, (b) row w_j has a_k in the k-th column if S_j contains attribute A_k, and (c) all other entries in T are distinct nondistinguished variables.

Step 2: (Tableau projection) For each i, $1\le i\le n$, take a projection of T onto R_i, denoted by $T[R_i]$, considering T as a relation on R. Let $T_i=T[R_i]$.

Step 3: (Chase and interactions) For each i, $1\le i\le n$, apply the FD-rule or JD-rule to T_i by D_i. Whenever a nondistinguished variable in some T_i is changed by applying FD-rule, change the nondistinduished variable appearing in other tableaux to the same variable. If any FD-rule or JD-rule or the above interaction cannot be applied, then let the final tableau be $\text{chase}_{D_i}(T_i)$ for each i, $1\le i\le n$.

Step 4: (Join of tableau) Take a join of $\text{chase}_{D_i}(T_i)$ for all i, denoted by $T'=\overset{n}{*}_{i=1}\text{chase}_{D_i}(T_i)$. If T' contains a row w such that w has distinguished variables in every column corresponding to an attribute in S, then output 'YES' else output 'NO'. Terminate.

Theorem 3 [15]

The above testing algorithm works correctly. That is, if the algorithm outputs 'YES', then the GJS $\overset{n}{*}_{i=1} R_i$ preserves the EJD $*[S_1,\ldots,S_m]$, and if it outputs 'NO', then the GJS does not preserve $*[S_1,\ldots,S_m]$.

Example 1

Let R_1=ABC and R_2=ACD be local schemes such that D_1={A → B, *[BC,AB]} and D_2={*[AC,AD]} hold in R_1 and R_2, respectively. First, let us consider the preservability of the JD *[ABC,AD] on $R_1 \cup R_2$ by R_1*R_2. Fig.5(a) shows the initial tableau T for *[ABC,AD], the projected tableaux T_1 and T_2, $chase_{D_1}(T_1)$ and $chase_{D_2}(T_2)$ and the join $chase_{D_1}(T_1)$*$chase_{D_2}(T_2)$, which are constructed by the above testing algorithm. Since we can obtain the row (a_1,a_2,a_3,a_4) in the join $chase_{D_1}(T_1)$*$chase_{D_2}(T_2)$, we conclude that the GJS R_1*R_2 preserves the JD *[ABC,AD]. That is, for any relations r_1 on R_1 and r_2 on R_2, such that $r_1 \in SAT(R_1,D_1)$ and $r_2 \in SAT(R_2,D_2)$, r_1*r_2 always satisfies the JD *[ABC,AD].

Next, let us consider the preservability of the JD *[ABD,BC]. Fig.5(b) shows the initial tableau T for *[ABD,BC] and several succeeding tableaux generated by the testing algorithm. We cannot obtain the row (a_1,a_2,a_3,a_4) in the final tableau $chase_{D_1}(T_1)$*$chase_{D_2}(T_2)$ and thus, R_1*R_2 cannot preserve the JD *[ABD,BC]. As described in the proof of Theorem 3, the tableaux $chase_{D_1}(T_1)$ and $chase_{D_2}(T_2)$ in Fig.5(b) can be considered to be relations on R_1 and R_2, respectively such that $chase_{D_1}(T_1) \in SAT(R_1,D_1)$, $chase_{D_2}(T_2) \in SAT(R_2,D_2)$ and the join of them does not satisfy the JD *[ABD,BC].

It is difficult to obtain the result only by the result by Beeri and Rissanen[10]. They assumed so called (pure) universal relation assumption (Fagin et al.[11]), which enforces that all relations must be projections of a large common relation at every time. Under this assumption, Beeri and Rissanen showed that for relation schemes R_i

$(i=1,\ldots,n)$ and a JD j defined on $\bigcup_{i=1}^{n} R_i$, j is preserved by $R_1*\ldots*R_n$ if and only if the union of D_i's and the JD $*[R_1,\ldots,R_n]$ logically implies j, where D_i denotes a set of FDs and JDs on R_i. By the original chase procedure shown in Fig.5(c), we can prove that

$$D_1 \cup D_2 \cup \{*[ABC,ACD]\} \models *[ABD,BC].$$

However, *[ABD,BC] cannot be preserved by R_1*R_2 as shown above.

From the testing algorithm and Theorem 3, we can obtain the following theorem immediately:

Theorem 4

Let $\{R_1,\ldots,R_n\}$ be an arbitrary set of local schemes such that a JD $*[S_1,\ldots,S_m]$ holds in some R_i. Then, the GJS $*_{i=1}^{n} R_i$ preserves the EJD $*[S_1,\ldots,S_m]$ if and only if for each j $(1\le j\le n,\ j\ne i)$, $\text{chase}_{D_j}(T_j)$ contains a row that has distinguished variables in all columns corresponding to attributes in $R_i \cap R_j$ (proof omitted).

Example 2

Let R_1=BCDEFG and R_2=ACEG be local schemes such that the JD *[BCDE,BCFG] holds in R_1 and other dependency constraints on R_1 and on R_2 are arbitrary. Fig.6 shows the initial tableau T for the EJD *[BCDE,BCFG], the projected tableaux T_1 and T_2, and a tableau T_1' obtained by applying JD-rule to T_1. Since $R_1 \cap R_2$=CEG, R_1*R_2 can preserve *[BCDE,BCFG] if and only if T_2 can be changed into a tableau which contains a row (b,a_3,a_5,a_7). Here, b denotes an arbitrary nondistinguished variable in T_2. The following conditions provides several sufficient conditions for *[BCDE,BCFG] to be preserved:

(a) BC → E holds in R_1 or BC → G holds in R_1.
(b) C → E holds in R_2 or C → G holds in R_2.
(c) *[CE,CG] holds in R_2.

We can easily prove that each of the conditions (b) and (c) enforces that T_2 has a row (b,a_3,a_5,a_7) in the chase process. The condition (a) seems to be a little strange since neither BC → E nor BC → G cannot be defined on R_2. If (a) holds (for example, BC → E holds), then we can obtain T_1'' from T_1' by applying FD-rule (BC → E). Then, at Step 3, the interaction occurs and the nondistinguished variable b_6 in T_2 is also changed into the distinguished variable a_5. Therefore, a row (b,a_3,a_5,a_7) is generated by condition (a).

From Theorem 4 and Example 2, we can obtain the following theorems immediately. This theorem provides a useful sufficient condition for a JD on R_1 to be preserved by $\{R_1,R_2\}$.

Theorem 5

Let $\{R_1,R_2\}$ be an arbitrary set of local schemes, where the JD $*[S_1,\ldots,S_m]$ holds in R_1. Then, R_1*R_2 preserves the EJD $*[S_1,\ldots,S_m]$ if the EJD $*[S_1 \cap R_2,\ldots,S_m \cap R_2]$ holds in R_2 (proof omitted).

Theorem 6

Assume that a JD $*[XY,XZ]$ holds in $R_1=XYZ$. R_1*R_2 preserves the EJD $*[XY,XZ]$ if at least one of the following conditions holds: (a) $*[XY \cap R_2, XZ \cap R_2]$ holds in R_2, (b) $X \rightarrow Y \cap R_2$ (or $X \rightarrow Z \cap R_2$) holds in R_1, (c) $X \cap R_2 \rightarrow Y \cap R_2$ (or $X \cap R_2 \rightarrow Z \cap R_2$) holds in R_2 (proof omitted).

Theorem 5 and Theorem 6 provide useful sufficient conditions when we need to rename some attributes so that some EJD may be preserved by a GJS. In the example shown previously, the EJD $*[\{DEPT,RES\}, \{DEPT,COM,DATE\}]$ cannot be preserved by the GJS R_1*R_2. If we rename attribute RES into DEV in R_2, then we can verify that the condition (a) of Theorem 6 holds. That is, $*[XY \cap R_2, XZ \cap R_2]$ is equal to $*[\emptyset,\{COM\}]$, which always holds. Here, X={DEPT}, Y={RES}, Z={COM,DATE} and R_2={DEV, COM, SOFT, LAN}.

RENAMING PLAN

Let j be a join dependency which holds in some R_k ($1 \le k \le n$). Assume that $R_1 * \ldots * R_n$ does not preserve j. The following renaming plan selects a set of attributes whose renaming makes j be preserved:

[Renaming Plan]

(1) Apply our testing algorithm to examine the preservability of j by the GJS $R_1 * \ldots * R_n$.

(2) Let T_i ($i \neq k$) be a tableau for R_i such that at least one row is changed by the tableau interaction until T_i is transformed into $chase_{D_i}(T_i)$. For each of these T_i's, let CANDIDATE(T_i) be a set of subsets of R_i such that each subset X of R_i satisfies the following conditions: (a) T_i has a row whose $R_k \cap (R_i - X)$ value consists of only distinguished variables, and (b) X is a minimal subset of R_i satisfying the condition (a).

(3) Let T_i ($i \neq k$) be a tableau for R_i such that any tableau interaction is not performed during its transformation into $chase_{D_i}(T_i)$. For each of these T_i's, let CANDIDATE(T_i) be a set of subsets of R_i such that each subset X satisfies the following conditions: (a) $chase_{D_i}(T_i)$ has a row whose $R_k \cap (R_i - X)$ value consists of only distinguished variables, and (b) X is a minimal subset of R_i satisfying the condition (a).

(4) Find a subset Y of R (the union of R_i's) that satisfies the following conditions: (a) For every CANDIDATE(T_i) ($i=1,\ldots,n$, $i \neq k$), Y contains at least one element of CANDIDATE(T_i), and (b) The number of attributes in Y is minimum.

(5) Rename the names of attributes in $R_i \cap Y$ for each relation scheme R_i ($i=1,\ldots,n$, $i \neq k$).

Example 3

Let R_1 and R_2 be LS's defined as follows:

R_1=ABCD, D_1={*[AB,ACD], *[ABC,BD]}

R_2=BCDE, D_2={*[BC,BDE]}.

As shown in Fig.7(a), *[AB,ACD] (a JD of R_1) cannot be preserved by R_1*R_2 since the $\text{chase}_{D_2}(T_2)$ does not contain a row (a_2,a_3,a_4,b). In this case, no interaction occurs at Step 3 in our testing algorithm, we obtain

Therefore, if we rename attribute B into B' of R_2, *[AB,ACD] is preserved by R_1*R_2. This is shown in Fig.7(b). Without renaming any attributes, the JD *[BC,BDE] of R_2 is preserved by R_1*R_2 (see Fig.7(c)). However, if we rename attribute B of R_2 into B' in order to preserve *[AB,ACD], we cannot preserve *[B'C,B'DE] of R_2 by R_1*R_2 (see Fig.7(d)). This is because any JD-rule cannot be applied to T_1. This example shows that some renaming plan to preserve a JD j_1 may cause other JD j_2 not to be preserved, where without any renaming, j_2 was preserved and j_1 was not preserved. Therefore, in order to preserve a set of JDs, we must select renaming plans so that any renaming plan does not violate the preservability of other JDs.

Fig.7(e) shows another example, which implies the necessity of taking a minimum cover of CANDIDATE(T_i')'s at (5) in our renaming plan if we wish to reduce the number of renamed attributes.

CONCLUDING REMARKS

In this paper, we discussed the logical integration problem of dependency constraints of local relational databases into a global view of a distributed database. When creating a global view by the bottom-up approach, some semantic constraints on a local database may be violated. It is important to calculate semantic constraints (especially, dependency constraints in the case of relational databases) of a global view that can be preserved by the global view. We have shown the relationship between a class of dependencies (FDs and EJDs) preserved by a global view and a class of dependencies of local relational databases. Here, any global view is assumed to be constructed by join, projection and attribute-renaming operations. A method is given for testing whether or not an embedded join dependency is preserved by a global view, where the global view is created by only join operations, and only FDs and JDs are given to each local database. By utlilizing this testing method, we can examine whether or not a given FD or a JD is preserved by a global view, which is constructed by join plus projection operations. We also discussed a method to remedy the violation of embedded join dependencies by renaming attributes. Our renaming plan makes an EJD (which was not preserved) to be preserved by renaming as few attributes as possible.

Recently, Klug[12] showed a method for calculating constraints (functional dependency and some non-dependency constraints) on relational expressions. The difference of our results from Klug's result is that we showed a technique for testing the preservability of not only functional but also embedded join dependencies. Furthermore, we also showed that the preservability problem of dependency constraints is important to create a logical scheme of a distributed database. As described in INTRODUCTION, our results will be useful to construct a heterogeneous distributed database since our approach is the bottom-up design approach. Recently, Motro et al.[17] and Masunaga[18] discussed a tool for integrating

(heterogeneous) independent databases into a global view. They suggested that Smith's generalization and aggregation concept[19] is useful to construct a global view. The relationship between their higher level abstraction facility and our dependency integration technique should be investigated as a future problem since their combination will provide a useful 'database integration tool'.

The results in this paper are useful not only to handle the logical integration problem, but also to discuss the dependency equivalence[13,14] of two centralized relational database schemes. Indeed, our study is motivated from the problem to test whether or not two relational database schemes can enforce the same dependency constraints, under the non-existence of the pure universal relation assumption[11]. This problem was partially treated in our previous work[13,14]. In this paper, we provide further results to cope with this problem.

Our testing method in this paper can be simplified (Step 4 can be eliminated). If we obtain a row with all distinguished variables in necessary columns for each tableau during the chase and interaction process, we can conclude that a given EJD is preserved and can terminate. It will be necessary to find more efficient algorithms to test the preservability of dependencies.

ACKNOWLEDGEMENTS

The authors are grateful to Professor Yajima and the colleagues in Yajima Laboratory at Kyoto University for their helpful discussions. This work is partly supported by the Science Foundation Grant of the Ministry of Education, Science and Culture of Japan. The work of K.Tanaka is also partly supported by Sakko-kai Foundation.

REFERENCES

[1] Rothnie,J.B.,Jr. et al, 'Introduction to a System for Distributed Databases (SDD-1)', ACMTODS, vol.5, no.1, pp.1-17, March 1980.

[2] Adiba,A., Andrade,J.M., Fernandez,E.B. and Nguyen,G.T., 'An Overview of the POLYPHEME Distributed Data Base Management System', Proc. IFIP Congress'80, pp.475-479, October 1980.

[3] Adiba,M., Caleca,J.Y. and Euzet,C., 'A Distributed Data Base System Using Logical Relational Machines', Proc. 4th International Conference on VLDB, pp.450-461, September 1978.

[4] Maier,D., Mendelzon,A.O. and Sagiv,Y., 'Testing Implications of Data Dependencies', ACMTODS, vol.4, no.4, pp.455-469, December 1979.

[5] Codd,E.F., 'Relational Completeness of Data Base Sublanguages', Proc. Courant Computer Science Symposium 6, Data Base Systems, May 1971.

[6] Codd,E.F., 'Further Normalization of the Data Base Relational Model', Proc. Courant Computer Science Symposium 6, Data Base Systems, May 1971.

[7] Rissanen,J., 'Theory of Relations for Databases - A Tutorial Survey', Lecture Notes in Computer Science 64, pp.536-551, September 1978.

[8] Sciore,E., 'A Complete Axiomatization of Full Join Dependencies', Dept. of EECS, Princeton University, Tech. Rep., TR-279, July 1979.

[9] Tanaka,K. and Kambayashi,Y., 'Testing of Join Dependency Preserving by a Modified Chase Method', to appear in Proc. 10th International Symposium on Mathematical Foundations of Computer Science'81, (Lecture Notes in Computer Science, Springer-Verlag), August 1981.

[10] Beeri,C. and Rissanen,J., 'Faithful Representations of Relational Database Schemes', IBM Res. Rep., RJ2722, January 1980.

[11] Fagin,R., Mendelzon,A.O. and Ullman,J.D., 'A Simplified Universal Relation Assumption and Its Properties', IBM Res. Rep., RJ2900, November 1980.

[12] Klug,A., 'Calculating Constraints on Relational Expressions', ACMTODS, vol.5, no.3, pp.260-290, September 1980.

[13] Kambayashi,Y., Tanaka,K. and Yajima,S., 'Problems of Relational Database Design', to appear in Lecture Notes in Computer Science, 'Database Engineering', Springer-Verlag, 1981.

[14] Tanaka,K., 'Studies on Logical Design Theory and Physical Storage Structures for Relational Databases', Doctoral Thesis, Faculty of Engineering, Kyoto University, January 1981.

[15] Tanaka,K. and Kambayashi,Y., 'Logical Integration of Locally Independent Relational Databases into a Distributed Database', to appear in Proc. 7th International Conference on VLDB, Cannes, France, September 1981.

[16] Smith,J.M. et al., 'Multibase - integrating heterogeneous distributed database systems', AFIPS NCC, pp.487-499, May 1981.

[17] Motro,A. and Buneman,P., 'Constructing Superviews', Proc. ACM-SIGMOD International Conference on Management of Data, pp.56-64, April 1981.

[18] 増永，野口 '分散データベースの統合問題について'. 電子通信学会 オートマトンと言語研究会技術報告 AL81-23, pp.51~58, 1981年6月

[19] Smith,J.M. and Smith,D.C.P., 'Database Abstractions: Aggregation and Generalization', ACMTODS, vol.2, no.2, pp.105-133, June 1977.

r_1

DEPT	RES	COM	DATE
CS	Tanaka	PC8000	1980/04/01
CS	Ono	PC8000	1980/04/01
CS	Tanaka	PDP11	1981/09/01
CS	Ono	PDP11	1981/09/01
EE	Yoshida	PC8000	1980/12/01
EE	Takaoka	PC8000	1980/12/01
EE	Yoshida	ACOS700	1980/04/01
EE	Takaoka	ACOS700	1980/04/01
IS	Kambayashi	PDP11	1976/04/01
IS	Kambayashi	Z8000	1981/08/01

Site 1

r_2

RES	COM	SOFT	LAN
Tanaka	ACOS700	Textual-DB	PL/I
Tanaka	PC8000	Wordprocessor	BASIC
Ono	ACOS700	Wordprocessor	FORTRAN
Ono	PDP11	Imageprocessor	FORTRAN
Kambayashi	PDP11	DBMS	C

Site 2

Fig.1 Example local relational databases r_1 and r_2.

$r_1 * r_2$

DEPT	RES	COM	DATE	SOFT	LAN
CS	Tanaka	PC8000	1980/04/01	Wordprocessor	BASIC
CS	Ono	PDP11	1981/09/01	Imageprocessor	FORTRAN
IS	Kambayashi	PDP11	1976/04/01	DBMS	C

Fig.2 A current instance $r_1 * r_2$ for a global join scheme $R_1 * R_2$.

$r_1 * r_2$

DEPT	RES	COM	DATE	DEV	SOFT	LAN
CS	Tanaka	PC8000	1980/04/01	Tanaka	Wordprocessor	BASIC
CS	Ono	PC8000	1980/04/01	Tanaka	Wordprocessor	BASIC
CS	Tanaka	PDP11	1981/09/01	Ono	Imageprocessor	FORTRAN
CS	Tanaka	PDP11	1981/09/01	Kambayashi	DBMS	C
CS	Ono	PDP11	1981/09/01	Ono	Imageprocessor	FORTRAN
CS	Ono	PDP11	1981/09/01	Kambayashi	DBMS	C
EE	Yoshida	PC8000	1980/12/01	Tanaka	Wordprocessor	BASIC
EE	Takaoka	PC8000	1980/12/01	Tanaka	Wordprocessor	BASIC
EE	Yoshida	ACOS700	1980/04/01	Tanaka	Textual-DB	PL/I
EE	Yoshida	ACOS700	1980/04/01	Ono	Wordprocessor	FORTRAN
EE	Takaoka	ACOS700	1980/04/01	Tanaka	Textual-DB	PL/I
EE	Takaoka	ACOS700	1980/04/01	Ono	Wordprocessor	FORTRAN
IS	Kambayashi	PDP11	1976/04/01	Ono	Imageprocessor	FORTRAN
IS	Kambayashi	PDP11	1976/04/01	Kambayashi	DBMS	C

Fig.3 A current instance $r_1 * r_2$ for a global join scheme $R_1 * R_2$, where R_1={DEPT, RES, COM, DATE} and R_2={DEV, COM, SOFT, LAN}. The EJD *[{DEPT,RES},{DEPT,COM,DATE}] holds in $r_1 * r_2$.

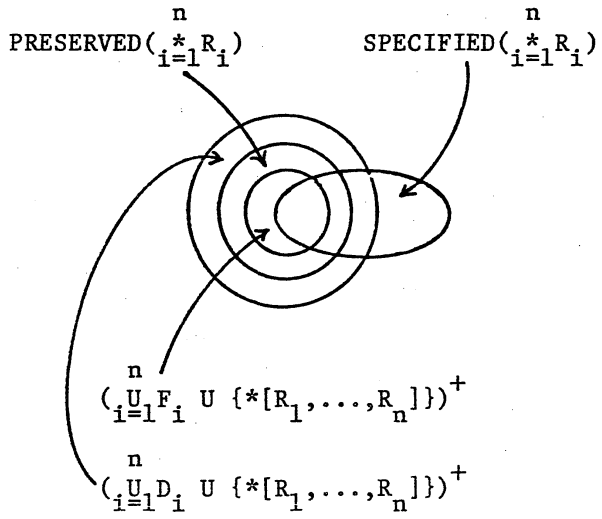
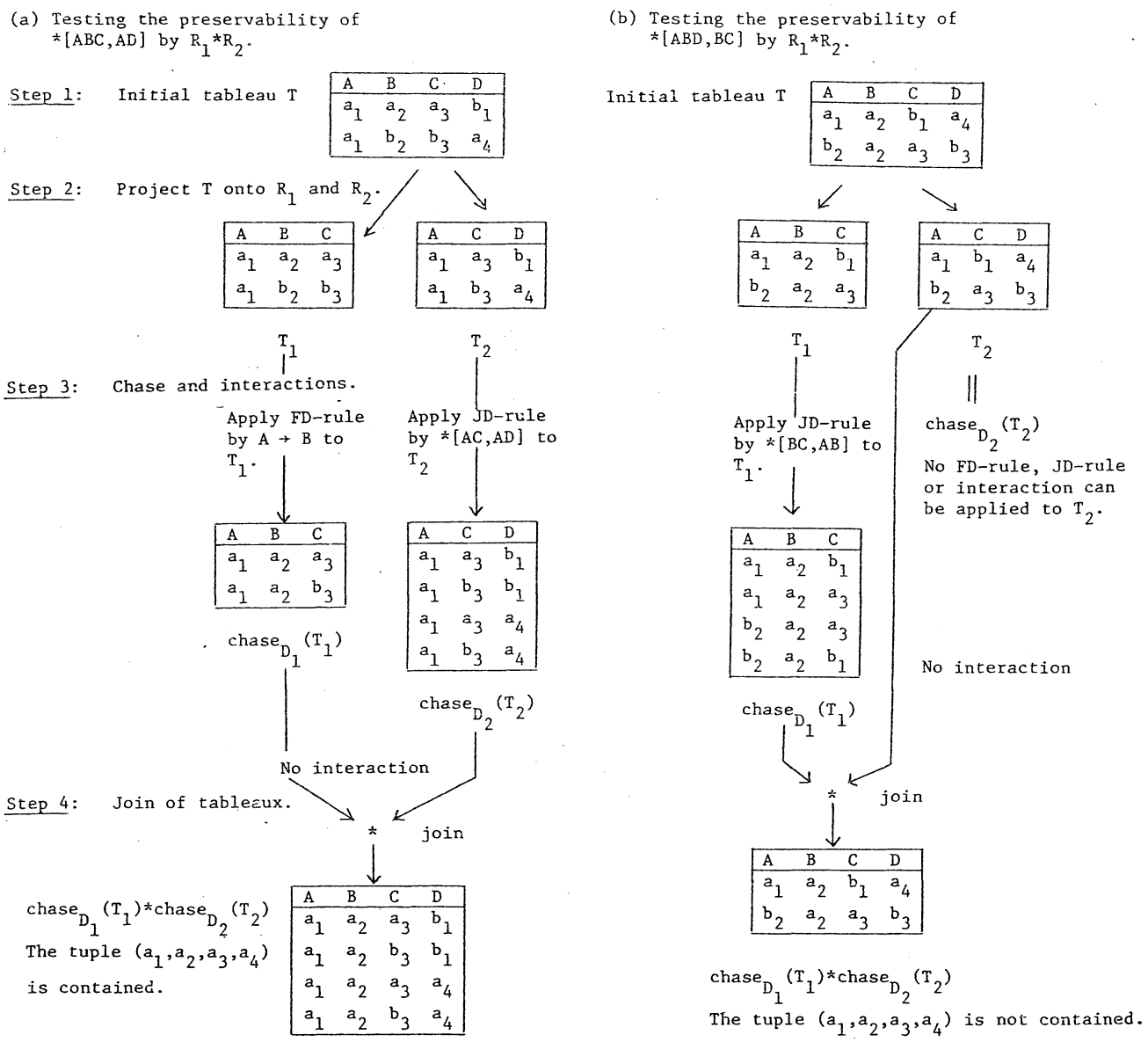


Fig.4 Relationships among a class of specified dependencies, a class of actually preserved dependencies and the constraints of local relational databases.
(Here, + denotes a closure of a set of FDs and EJDs.)



(c) Ordinary chase process using the result by Beeri and Rissanen[10].

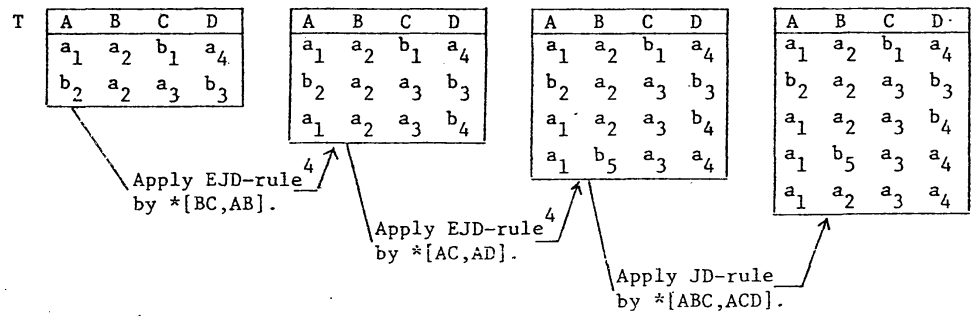


Fig.5 Testing the preservability of EJDs for the GJS R_1*R_2, where R_1=ABC and R_2=ACD.

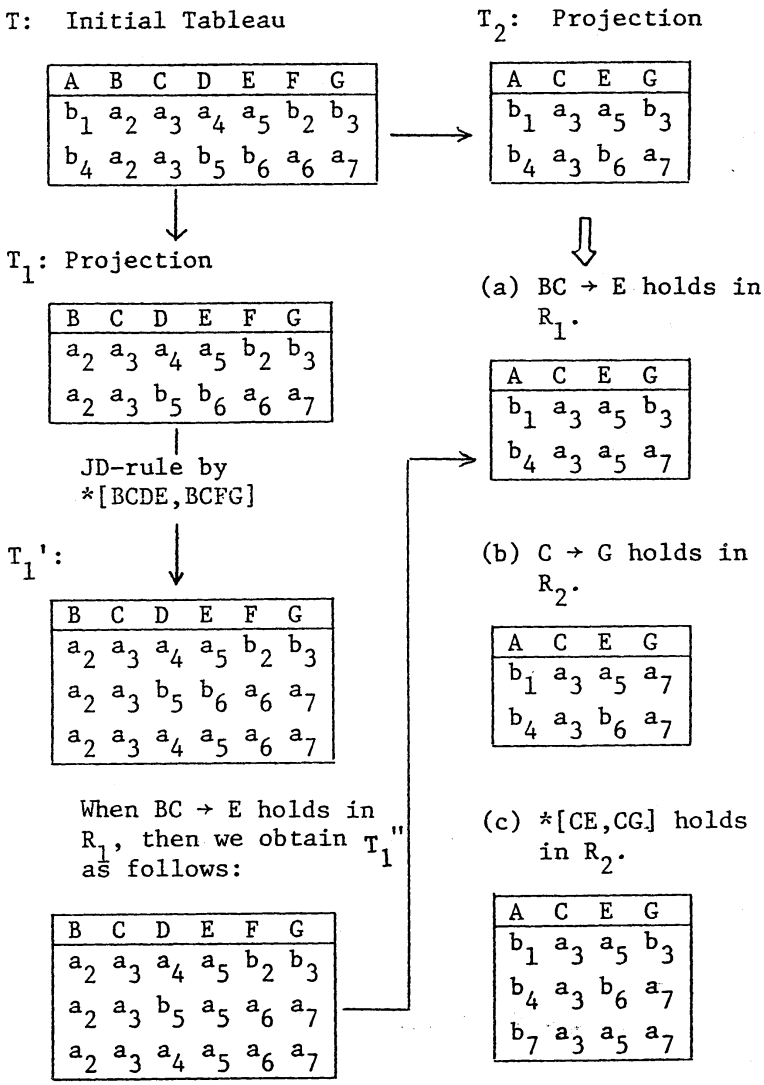


Fig.6 Conditions for the EJD *[BCDE,BCFG] to be preserved by $R_1 * R_2$.

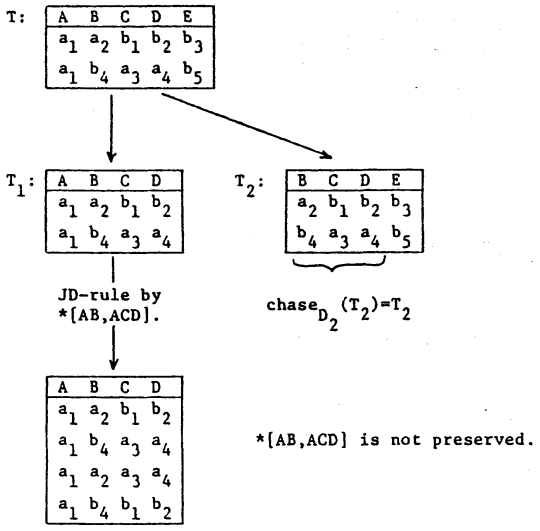


Fig.7 (a) Preservability of *[AB,ACD].

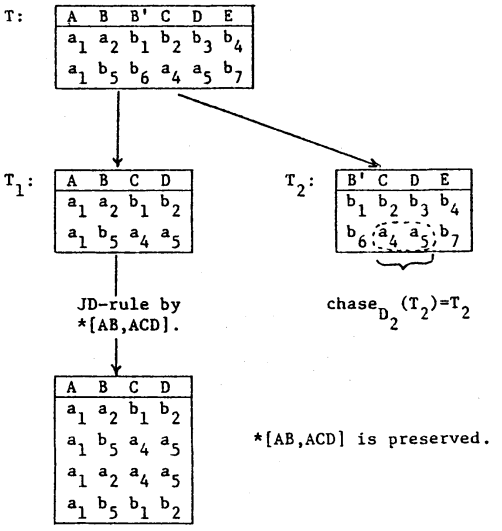


Fig.7 (b) Preservability test of *[AB,ACD] after renaming attribute B of R_2 into B'.

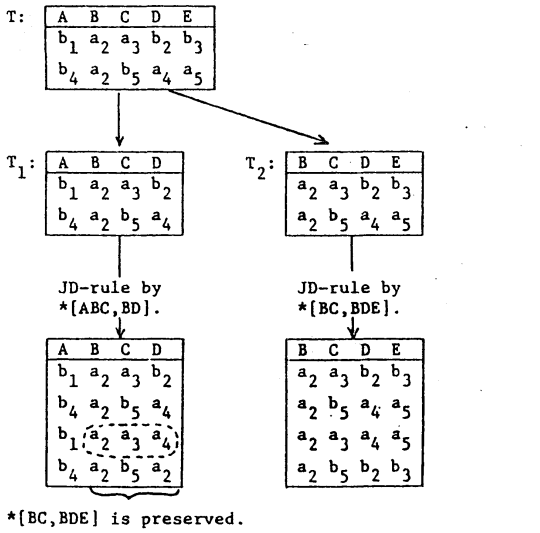


Fig.7 (c) Preservability test of *[BC,BDE].

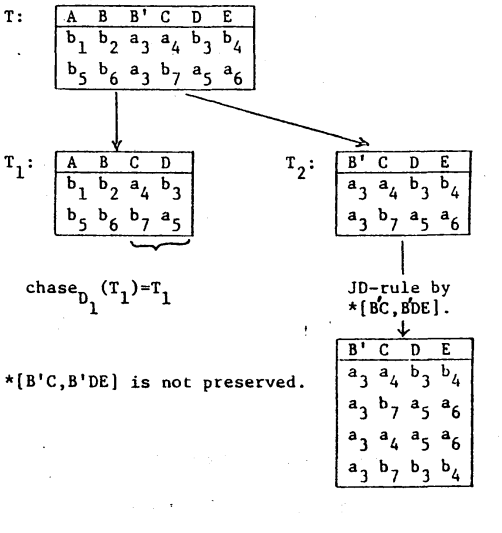


Fig.7 (d) Preservability test of *[B'C,B'DE].

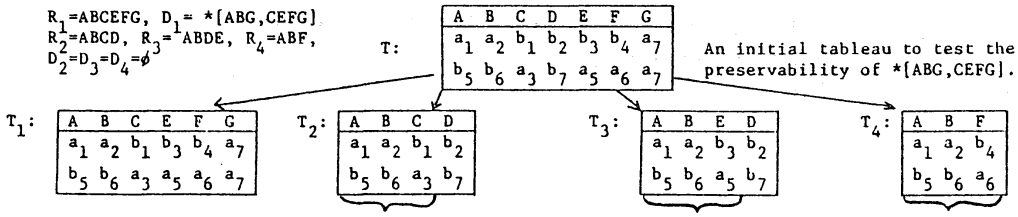


Fig.7 (e) An example to show the necessity of taking a minimum cover at (5) in our renaming plan.