Independence-reducible Database Schemes

E.P.F. Chan and H.J. Hernández

Research Report CS-88-18
April 1988

Independence-reducible Database Schemes[†]

Edward P.F. Chan[‡]

Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

Héctor J. Hernández[§]

Department of Computer Science
Texas A&M University
College Station, Texas
USA 77843–3112

ABSTRACT

We define a generalization of Sagiv-independent database schemes, called (key-equivalent) independence-reducible schemes, and show that it is highly desirable with respect to query answering and constraint enforcement. To demonstrate the class of schemes identified is rather general, we prove that it contains a superset of all previously known classes of cover embedding BCNF database schemes with similar properties. An efficient algorithm is found which recognizes exactly this class of database schemes. Independence-reducible database schemes properly contain a class of constant-time-maintainable database schemes. A condition is found which characterizes this class of schemes and the condition can be tested efficiently. Throughout, it is assumed that a cover of the functional dependencies is embedded in the database scheme in the form of key dependencies.

April 4, 1988

[†]A preliminary version of this paper appears in the Proceedings of the 7^{th} ACM Symposium on Principles of Database Systems, Austin, 1988, pp. 163-173.

[‡]The work of E. Chan was supported by a grant from Natural Sciences and Engineering Research Council of Canada.

[§]The work of H. Hernández was supported by a grant from Texas A&M University.

Independence-reducible Database Schemes[†]

Edward P.F. Chan[‡]

Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

Héctor J. Hernández[§]

Department of Computer Science
Texas A&M University
College Station, Texas
USA 77843—3112

1. Introduction

The design theory began with the pioneering work of Codd [Cod1][Cod2]. Codd observed that in the presence of functional dependencies, updating a relation may result in certain problems which are widely known as the *update anomalies* [Cod2]. Codd suggested normalization as a way to separate independent facts into different relations and to reduce logical data duplication. Since then, much research on design theory centered around this problem. Various normal forms as well as other desirable properties associated with normalization have been proposed [ABU][Bern][BDB][Cod1][Cod2][Cod3][F1][F2][Z]. Under a different setting, the problem of how to store independent facts into separate relations was later studied by other authors [CM1][GY][IIK][S1][S2].

Query answering is an important function in a database system. Consequently, designing a database scheme that facilitates efficient query answering is highly desirable. Recent work on acyclicity [BFMY][F3][DM][Y1] addressed the problem of what types of database structures allow efficient query answering in the presence of the full join dependency. With functional and full join dependencies, several classes of database schemes with such a desirable property were identified [AC1][CH1][CM2][GY][IIK][MRW][S3]. The notion of boundedness [GM][MUV] was used in the latter work as the evaluation criterion.

Constraints are logical restrictions imposed on a database. Ensuring a database satisfies the constraints is costly in general. The idea of independence was the first attempt in characterizing when a database scheme allows efficient solution to the constraint enforcement problem. Informally, a database scheme is independent if each relation in a state satisfies a specific set of constraints implies the state is globally consistent [S1][S2][GY][IIK]. Constant-time-maintainability is a generalization of independence in which the constraint enforcement problem could be solved in time independent of the state size [GW].

In this paper, we are interested in identifying a rather general class of schemes which is desirable with respect to query answering and constraint enforcement. In view of the importance of key dependencies [Bern][S1][S2], we assume throughout a cover of functional dependencies is embedded in a database in the form of keys. There are two classes of schemes which were proven to be highly desirable with respect to query answering and constraint enforcement when a set of key dependencies is

considered. They are the class of independent schemes [S1][S2] and the class of γ-acyclic schemes [CH1]. However, there are database schemes which possess the same desirable properties but do not fall into the above classes of schemes.

Example 1: Let us consider a university database in which a course could be taught by more than one teacher. Let the database scheme be $\mathbf{R} = \{R_1(HRC), R_2(HTR), R_3(HTC), R_4(CSG), R_5(HSR)\}$, where C = *course*, T = *teacher*, H = *hour*, R = *room*, S = *student* and G = *grade*. The sets of (candidate) keys for R_1 to R_5 are $\{HR\}$, $\{HT, HR\}$, $\{HT\}$, $\{CS\}$ and $\{HS\}$ respectively. The set of constraints is the set of key dependencies implied by the keys embedded in **R**. **R** is neither independent nor γ-acyclic. Because of the interaction between the relation schemes and the constraints, it is not obvious if **R** is bounded. In fact, by the results in this paper, not only can we show that **R** is bounded, but also it is constant-time-maintainable. So this scheme is indeed highly desirable with respect to query answering and constraint enforcement. □

Let us consider the following database scheme $\mathbf{S} = \{S_1(HRCT), S_2(CSG), S_3(HSR)\}$. The sets of keys for relation schemes S_1 to S_3 are $\{HR, HT\}$, $\{CS\}$ and $\{HS\}$ respectively. So the set of key dependencies implied by keys in **S** is identical to the set of key dependencies implied by keys in **R** in Example 1. By a result in [S2], **S** is independent with respect to the set of key dependencies. Suppose that we allow unknown but existing nulls in a relation (null values may even appear in a key), every state **r** on **R** can be transformed into a state **s** on **S**. This can simply be done by extending tuples from R_1, R_2 and R_3 to S_1 with distinct nulls, relations on R_4 and R_5 are simply relations on S_2 and S_3 respectively. Since **S** is independent, it is not difficult to see that **r** is consistent with respect to the set of key dependencies exactly when the substate on R_1, R_2 and R_3, and the relations on R_4 and R_5 all satisfy their respective embedded key dependencies. Although **R** is not an independent scheme, verification of consistency of a state can still be carried out by inspecting certain subset of relations in the state. In this sense, **R** in Example 1 can be considered as an instance of a more general kind of independent schemes. This type of schemes is the object of study in this paper and they will be called (key-equivalent) independence-reducible schemes.

In this paper, we first show that a class of database schemes, called key-equivalent database schemes, is bounded with respect to a set of key dependencies. With this result, we then show that the constraint enforcement problem for this class of schemes can be solved incrementally via predetermined relational expressions. After that, we propose a technique called independence-reducibility to show that a larger class of cover embedding schemes, which we call key-equivalent independence-reducible database schemes, or independence-reducible database schemes for short, also possesses all the desirable properties enjoyed by key-equivalent database schemes. Hence the class of schemes identified is highly desirable with respect to query answering and constraint enforcement. To demonstrate that the class of schemes identified is rather general, we prove that the class of independence-reducible database schemes contains a superset of all previously known classes of cover embedding BCNF database schemes that with similar properties [S1][S2][CH1]. Independence-reducible schemes properly contain a class of constant-time-maintainable database schemes and an efficient algorithm is found for testing when an independence-reducible database scheme is constant-time-

maintainable.

In Section 2, we give most of the definitions needed in this paper. In Section 3, we define the class of key-equivalent database schemes and show that this class of schemes is bounded and algebraic-maintainable. Algebraic-maintainability is a generalization of constant-time-maintainability. We then give a condition that exactly determines when a key-equivalent database scheme is constant-time-maintainable, and show that it can be tested efficiently. In Section 4, we define the class of independence-reducible database schemes and prove its properties. We show that independence-reducible schemes are bounded, algebraic-maintainable and closed under an augmentation operation. Algorithms for computing total projections and for determining if an updated state is consistent are also given. In Section 5, we find an efficient algorithm which recognizes exactly the class of independence-reducible database schemes. We also prove that this class of schemes contains a superset of all previously known classes of cover embedding BCNF database schemes which are bounded and constant-time-maintainable. Finally, we show that the characterization of constant-time-maintainability for key-equivalent database schemes is applicable to independence-reducible schemes. In Section 6, we summarize the results in this paper.

2. Definitions and Notation

In this section, we give most of the notation required for the rest of this paper.

2.1. Basic Definitions

A *partition* of a set S is a collection of nonempty subsets of S such that elements in the collection are pairwise disjoint and the union of the collection is S. Each subset in the collection is called a *block*. Two sets are *incomparable* if neither one is a subset of the other.

Following standard notation [Ma][U], we fix a finite set of attributes $U=\{A_1, \ldots, A_n\}$ and call it the *universe*. With each A_i we associate a set of constants called its *domain*, denoted by $dom(A_i)$. Domains for different attributes are assumed to be disjoint. A *relation scheme* R is a subset of U. A *database scheme* $\mathbf{R}=\{R_1, \ldots, R_k\}$ is a collection of relation schemes such that the union of the R_i's is U. A *tuple* defined on $R=\{A_1, \ldots, A_j\}$ is a function μ that maps each A_i to a value, $1 \leq i \leq j$. The value can either be a constant, from $dom(A_i)$, or a *variable* taken from an infinite set of uninterpreted symbols. If μ is a tuple on R and X is a subset of R, $\mu[X]$ denotes the *restriction* of μ to X. We say $\mu[X]$ is *total* if $\mu[A_i]$ is a constant, for all $A_i \in X$. Let $\pi^{\downarrow}$ be the *restricted projection operator* and be defined as $\pi^{\downarrow}_X(I) = \{t[X] \mid t \in I \text{ and } t[X] \text{ is total}\}$, where I is a set of tuples. Let π denote the usual projection operator. A *relation* on R is a set of tuples defined on R such that every tuple is total. A *database state* is a function $\mathbf{r}$ that maps each relation scheme $R_i \in \mathbf{R}$ to a relation on R_i. We write $\mathbf{r}=<r(R_1), \ldots, r(R_k)>=<r_1, \ldots, r_k>$. Let $\mathbf{r}=<r_1, \ldots, r_k>$ and $\mathbf{s}=<s_1, \ldots, s_k>$ be two states on $\mathbf{R}$. Then $\mathbf{s} - \mathbf{r}=<s_1-r_1, \ldots, s_k-r_k>$, and $\mathbf{s} \cup \mathbf{r} = <s_1 \cup r_1, \ldots, s_k \cup r_k>$.

2.2. Tableaux

A *tableau* is a set of tuples or *rows* defined on U [ASU]. Each column of a tableau corresponds to an attribute in U. The domain of the i^{th} column of the tableau, corresponding to an attribute A_i, consists of the *distinguished variable (dv)* a_i, a set of countable many *nondistinguished variables (ndv's)* $\{b_{ij}\}$ and constants taken from the domain of A_i. No variables can appear in two different columns in a tableau. A tableau can contain redundant rows. A tableau is said to be *minimized* if no proper subset is equivalent to the tableau. A detailed discussion on equivalence and minimization of tableaux can be found in [ASU].

Given a database state $\mathbf{r}=<r_1, \ldots, r_k>$, we define a tableau T_r on U and call it *the tableau for database state* $\mathbf{r}$: For each relation $r_i \in \mathbf{r}$, and for each tuple $t \in r_i$, there is a row s in T_r corresponding to it. The row s is said to *originate* from r_i (or R_i) and is defined as follows:

- $s[R_i] = t$;
- $s[A] = b_{ij}$, b_{ij} is a ndv that appears nowhere else in T_r, for all $A \in U - R_i$.

The *tableau for a database scheme* $\mathbf{R}=\{R_1, \ldots, R_k\}$, denoted $T_{\mathbf{R}}$, is a tableau of k rows where each row corresponds to exactly one R_i in $\mathbf{R}$ [ABU][ASU]. The components of the rows in $T_{\mathbf{R}}$ are defined as follows: If t_i is the row in $T_{\mathbf{R}}$ for relation scheme R_i, $t_i[A_j] = a_j$, if $A_j \in R_i$, else $t_i[A_j] = b_{ij}$.

2.3. Functional Dependencies and Chasing

The kind of constraints considered here is functional dependencies (fd's) [Ma][U]. Associated with each fd is an *fd–rule*. Given a tableau T and a set of fd's, we can use the fd-rules to infer additional information by equating symbols of T. These transformation rules are defined as follows and their properties are described in [MMS]:

> *fd–rule*: for each fd $X \rightarrow A$, there is an fd-rule corresponding to it. Suppose tableau T has rows t_1, t_2 that agree in all X-columns. Let v_1, v_2 be the values in the A-column of t_1, t_2 respectively. Furthermore, assume $v_1 \neq v_2$. Applying the fd-rule corresponding to $X \rightarrow A$ to rows t_1, t_2 of T yields a transformed tableau T'. T' is the same as T except v_1, v_2 are renamed as follows. If one of v_1 or v_2 is a dv (or a constant) and the other is not, then rename the other by the dv (or the constant respectively). If both are ndv's, then rename the variable with the higher subscript to be the variable with the lower subscript. If both are distinct constants, the result of applying the rule is usually defined to be the empty tableau and an *inconsistency* is said to be found.

Let $\chi = \chi_1 \cdots \chi_n$ be a sequence of transformations, then $\chi(T)$ denotes the application of transformations χ to the tableau T. $\chi(T)$ is defined as $\chi_n(\cdots(\chi_1(T))\cdots)$. Suppose F is a set of fd's, $CHASE_F(T)$ means that we apply the corresponding fd-rules exhaustively to T.

Given a set of dependencies F, there are additional dependencies implied by this set in the sense that any relation that satisfies this set must also satisfy the additional dependencies. The set of dependencies that is logically implied by F is the *closure* of F, denoted by F^+. Two sets of fd's F and G are *equivalent*, or F is a *cover* of G, if $F^+ = G^+$. Given a set of attributes X, the *closure* of X *with respect to (wrt)* F,

denoted by X_F^+ (or X^+ if F is clearly understood), is the set of attributes $\{A \mid X \rightarrow A \in F^+\}$.

An fd $X \rightarrow A$ is said to be *embedded* in a relation scheme R if $XA \subseteq R$. The projection of a set of fd's F onto R_i, denoted by $F^+ \mid R_i$, is the set of *projected fd's* $X \rightarrow A \in F^+$ such that XA is embedded in R_i. A database scheme $\mathbf{R}$ is said to be *cover embedding* wrt a set of fd's F if there exists a cover G of F such that for each fd $X \rightarrow A \in G$, $X \rightarrow A$ is embedded in some $R_i \in \mathbf{R}$. G is said to be an *embedded cover* of $\mathbf{R}$. If $F^+ = G^+$ then $CHASE_F(T) = CHASE_G(T)$, for any tableau T on U [MMS].

Given a set of fd's F, a nonempty subset K of a relation scheme R is called a *candidate key*, or simply a *key* of R if $K \rightarrow R \in F^+$ and no proper subset of K has this property. $X \subseteq R$ is a *superkey* of R if X contains a key of R. If K is a key of R and $A \in R - K$, we say that $K \rightarrow A$ is a *key dependency* embedded in R. F is a set of key dependencies *(embedded) in* R if F is equivalent to the set $\{F \rightarrow A \mid F \rightarrow A$ is a key dependency in $R\}$. F is a set of key dependencies *(embedded) in* a database scheme $\mathbf{R}$ if F is equivalent to the set $\cup\{F_i \mid F_i$ is a set of key dependencies in R_i, $R_i \in \mathbf{R}\}$. A database scheme $\mathbf{R}$ is in *Boyce-Codd Normal Form (BCNF)* wrt a set of fd's F if for all nontrivial $X \rightarrow Y \in F^+$ embedded in some R_i, $R_i \in \mathbf{R}$, X is a superkey of R_i. If $\mathbf{R}$ is a cover embedding database scheme wrt a set of embedded key dependencies, we assume the set is explicitly given. A database scheme $\mathbf{R}$ is *lossless* wrt F if $CHASE_F(T_\mathbf{R})$ has a row of all dv's. $\mathbf{S} \subseteq \mathbf{R}$ is said to be a *lossless subset of* $\mathbf{R}$ *covering* X if $\cup\mathbf{S} \supseteq X$ and $\mathbf{S}$ is lossless wrt the fd's embedded in $\mathbf{S}$.

2.4. Hypergraphs for Database Schemes

A *hypergraph* is a pair $H = \langle V, E \rangle$, where V is a set of *nodes* and E is a collection of nonempty subsets of V called *edges* [B]. Any subset of E in H forms a *subhypergraph* of H.

Given a database scheme $\mathbf{R}$, its hypergraph, denoted by $H_\mathbf{R}$, has U as its set of nodes, and $\mathbf{R}$ as its set of edges. In this paper, we are interested in γ-acyclicity [DM][F3]. Following [ADM][F3], we give below the required terminology of hypergraphs used in this paper.

Let $H = \langle V, E \rangle$ be a hypergraph. A *path* from x_1 (E_1) to x_m (E_m) is a sequence $\langle E_1, E_2, \ldots, E_m \rangle$ such that:

- $x_1 \in E_1$ and $x_m \in E_m$;
- $E_1, E_2, \ldots, E_m$ are edges in E, $m \geq 1$;
- $E_k \cap E_{k+1} \neq \emptyset$, for $k = 1, 2, \ldots, m-1$;
- no proper subsequence of it satisfies the above properties.

Two nodes (edges) are *connected* if there exists a path from one to the other. $H = \langle V, E \rangle$ is *connected* if every pair of nodes (edges) in H are connected.

Given a family of sets $E = \{E_1, \ldots, E_n\}$, *Bachman(E)* is defined as follows:

- if $E_i \in E$, then $E_i \in$ Bachman(E);

- if X and Y are in Bachman(E), then $X \cap Y$ is in Bachman(E);
- nothing else is in Bachman(E).

A family of sets $\{W_1, \ldots, W_m\}$ is *connected* if the hypergraph $H = < \bigcup_{i=1}^{m} W_i, \bigcup_{i=1}^{m} \{W_i\} >$ is connected. Let **R** be a database scheme. A connected set $V = \{V_1, \ldots, V_m\} \subseteq$ Bachman(**R**) is a *unique minimal connection (u.m.c.) among* $X \subseteq U$, if

- $\bigcup_{i=1}^{m} V_i \supseteq X$, and
- for every connected subset $\{W_1, \ldots, W_k\}$ of Bachman(**R**) such that $\bigcup_{i=1}^{k} W_i \supseteq X$, there exists $\{W_{i_1}, \ldots, W_{i_m}\} \subseteq \{W_1, \ldots, W_k\}$ such that $W_{i_j} \supseteq V_j$, for $1 \leq j \leq m$.

There are several efficient methods of finding the u.m.c. [BBSK][C2][Y2]. The following result concerning u.m.c.'s and γ-acyclic hypergraphs is stated in [F3][Y2], and recently proven in [BBSK].

Theorem 2.1: Let **R** be a database scheme and be connected. **R** is γ-acyclic if and only if **R** has a u.m.c. among X, for any $X \subseteq U$.

2.5. Weak Instances and Boundedness

Let **r** be a state for a database scheme $\mathbf{R} = \{R_1, \ldots, R_k\}$. Let I be a relation defined on U. Then I is a *weak instance* for **r** wrt a set of dependencies F if

- $\pi_{R_i}(I) \supseteq r_i$, for each $1 \leq i \leq k$;
- I satisfies F.

A database state **r** is said to be *consistent* wrt a set of dependencies F if a weak instance exists for the state wrt F [GMV][H2]. It has been shown that $CHASE_F(T_r)$ is nonempty if and only if **r** is a nonempty consistent state [H2]. $CHASE_F(T_r)$ is called *the representative instance for state* **r**. The *X-total projection* of the representative instance for **r**, denoted $[X]$, is $\pi\!\downarrow_X(CHASE_F(T_r))$. This model is known as the *weak instance model* and has been used by various authors as a formal way to study information content in a database. See for example, [AC1][AD][C1][CA][CKS][CH1][CM2][GW][GY][IIK][M][MUV][S1][Y1].

A database scheme **R** is *bounded* wrt F if every total tuple t in the representative instance of any consistent state **r** of **R** wrt F can be obtained in at most k fd-rule applications starting from T_r, for some constant $k \geq 0$ [GM][MUV]. It has been shown that a database scheme is bounded wrt F if any total tuple in the representative instance for a consistent state can be computed by a predetermined relational expression [GM][MUV].

The largest classes of database schemes known to be bounded wrt fd's are the class of independent database schemes [AC1][C1][IIK][MRW][S3] and the class of γ-acyclic cover embedding BCNF database schemes [CH1]. Chan and Hernández investigated how to generate bounded database schemes incrementally [CH2]. Sagiv studied the problem of computing total projections on the universe in the presence of full implicational dependencies. He obtained a characterization for computing such a total

projection when a set of tuple generating dependencies is considered [S4]. Recently, Chan and Hernández found a general sufficient condition for unboundedness when fd's are considered [CH3].

2.6. Extension Joins and Sequential Joins

An *extension join* is of the form $E_1 \bowtie \pi_{(R_1 \cap R_2) \cup Y}(E_2)$, where E_1 and E_2 are either relation schemes or extension joins defined on R_1 and R_2 respectively, $Y \subseteq R_2 - R_1$ and $R_1 \cap R_2 \rightarrow Y \in F^+$.

Let E be a join expression on $\mathbf{R}$. If E is of the form $(((R_1 \bowtie R_2) \bowtie R_3) \cdots \bowtie R_n)$, where $R_1, \ldots, R_n$ is an ordering of distinct members of $\mathbf{R}$, then we say that E is *sequential* [F3]. Intuitively a sequential join corresponds to first joining R_1 and R_2, then joining the result with R_3 and so on.

2.7. Independence, Constant-time-maintainability and Algebraic-maintainability

The *maintenance problem* (for database states) of $\mathbf{R}$ wrt F is the following decision problem: Let $\mathbf{r}$ be a consistent state of a database scheme $\mathbf{R}$ wrt F and assume we insert a tuple t into $r_p \in \mathbf{r}$. Is $\mathbf{r}' = \mathbf{r} \cup \{t\}$ a consistent state of $\mathbf{R}$ wrt F?

We say that $<\mathbf{r}, t>$ above is an *instance* of the maintenance problem of $\mathbf{R}$ wrt F. An algorithm is said to *solve* the maintenance problem of $\mathbf{R}$ wrt F if the algorithm correctly determines if an instance of the maintenance problem is consistent or not [GY].

In view of the importance of the maintenance problem, designing a database scheme that allows efficient solution to the problem is essential. The first such class of desirable schemes proposed is called independent schemes and is defined as follows.

Let the set of consistent states for a database scheme $\mathbf{R}$ wrt a set of dependencies F be denoted by $\mathrm{WSAT}(\mathbf{R}, F) = \{\mathbf{r} \mid \mathbf{r}$ is a state of $\mathbf{R}$ and is consistent wrt $F\}$. The *locally consistent states* of $\mathbf{R}$ are elements of the set $\mathrm{LSAT}(\mathbf{R}, F) = \{\mathbf{r} \mid r_i$ satisfies $F^+|R_i$, for each $r_i \in \mathbf{r}\}$. That is, $\mathbf{r}$ is locally consistent if no relation $r_i \in \mathbf{r}$ violates any projected dependencies. A database scheme $\mathbf{R}$ is said to be *independent* wrt a set of dependencies F if $\mathrm{LSAT}(\mathbf{R}, F) = \mathrm{WSAT}(\mathbf{R}, F)$. The independent schemes were proposed and investigated in [GY][IIK][S1][S3]. The constraints that they considered include fd's and the full join dependency. Recently, this class of schemes was studied in the presence of fd's and inclusion dependencies [AC2].

Suppose $\mathbf{R} = \{R_1, \ldots, R_k\}$ is cover embedding wrt $F = F_1 \cup \cdots \cup F_k$, where F_i is a set of key dependencies embedded in $R_i \in \mathbf{R}$, for all $1 \leq i \leq k$. Then independence is characterized by a condition called the *uniqueness* condition. $\mathbf{R}$ is said to satisfy the uniqueness condition if for all R_i, R_j in $\mathbf{R}$, $R_i \neq R_j$, $(R_i)^+_{F - F_j}$ does not contain a key dependency embedded in R_j [S1][S2].

Recently, Graham and Wang [GW] generalized the concept of independent schemes and defined a class of database schemes called *constant-time-maintainable (ctm)* schemes. Informally, a database scheme is ctm if there is an algorithm which solves the maintenance problem by retrieving 'exactly' those tuples in the state that need to be examined in determining if the updated state is consistent. Moreover, the

number of tuples retrieved is 'small'. We now give a definition of constant-time-maintainability as follows.

Let Φ be a conjunction of equality formulae of the form $A = 'a'$, where $A \in R_i \in \mathbf{R}$ and $a \in dom(A)$. Φ is said to be a *conjunctive* formula. A selection $\sigma_\Phi(E)$ is a *conjunctive* selection if Φ is a conjunctive formula. Define $CST(\Phi)$ as $\{a \mid A = 'a'$ is an equality formula in $\Phi\}$. Let t be a tuple, then $CST(t)$ is the set of constants in t. Let $\{t_1, \ldots, t_n\}$ be a set of tuples, then $CST(\{t_1, \ldots, t_n\}) = CST(t_1) \bigcup \cdots \bigcup CST(t_n)$. Let E be a relational expression on $\mathbf{R}$. E is said to be *single-tuple* if for any consistent state $\mathbf{r}$, $E(\mathbf{r})$ always returns a set containing at most one tuple. Let $\chi = \sigma_{\Phi_1}(E_1), \ldots, \sigma_{\Phi_n}(E_n)$, $n \geq 0$, be a sequence of single-tuple conjunctive selections on $\mathbf{R}$. The sequence χ is said to *define on* an instance $<\mathbf{r}, t>$ if $CST(\Phi_1) \subseteq CST(t)$ and for all $1 < i \leq n$, $CST(\Phi_i) \subseteq CST(\{t\} \bigcup \sigma_{\Phi_1}(E_1(\mathbf{r})) \bigcup \cdots \bigcup \sigma_{\Phi_{i-1}}(E_{i-1}(\mathbf{r})))$. A database scheme $\mathbf{R}$ is *ctm* wrt F if there is an algorithm that correctly determines if an instance $<\mathbf{r}, t>$ is consistent by examining the tuple t and tuples generated from applying a sequence $\sigma_{\Phi_1}(\pi_{X_1}(R_1)), \ldots, \sigma_{\Phi_n}(\pi_{X_n}(R_n))$ of single-tuple conjunctive selections on $<\mathbf{r}, t>$ to $\mathbf{r}$, where for all $1 \leq i \leq n$, $X_i \subseteq R_i$, $R_i \in \mathbf{R}$, and n is dependent only on $\mathbf{R}$ and F.

The class of ctm database schemes is desirable wrt constraint enforcement since there is an algorithm which solves the maintenance problem by examining a number of tuples which is independent of the state size. Furthermore the set of tuples can be computed easily. A more general class of database schemes that allows efficient solution to the maintenance problem is defined as follows.

A database scheme $\mathbf{R}$ is *algebraic-maintainable* wrt F if there is an algorithm which correctly determines if an instance $<\mathbf{r}, t>$ is consistent by examining the tuple t and tuples generated from applying a sequence of single-tuple conjunctive selections $\sigma_{\Phi_1}(E_1), \ldots, \sigma_{\Phi_n}(E_n)$ on $<\mathbf{r}, t>$ to $\mathbf{r}$, where for all $1 \leq i \leq n$, E_i is an expression on $\mathbf{R}$, and the sizes of the expressions and the sequence are dependent only on $\mathbf{R}$ and F. We first show by an example that not every database scheme is algebraic-maintainable.

Example 2: Let $\mathbf{R} = \{R_1(AB), R_2(BC), R_3(AC)\}$ and $F = \{A \rightarrow C, B \rightarrow C\}$. Consider the following state tableau for a consistent state $\mathbf{r}$ on $\mathbf{R}$. The *TAG*-column indicates from which relation a tuple originates.

A	B	C	*TAG*
a_0		c_0	R_3
a_0	b_0		R_1
a_1	b_0		R_1
a_1	b_1		R_1
.			.
.			.
.			.
a_n	b_n		R_1

Now suppose we insert a tuple $<a_n, c_n>$ into r_3, $c_o \neq c_n$. Since any proper substate of $\mathbf{r}$ with the tuple $<a_n, c_n>$ is consistent, all tuples in $\mathbf{r}$ need to be retrieved to show the inconsistency of the updated state. This implies that the size of the sequence of single-tuple conjunctive selections is dependent on the state size. Hence $\mathbf{R}$ is not

algebraic-maintainable wrt F. □

By definitions, an independent scheme is a ctm scheme, and a ctm scheme is an algebraic-maintainable scheme. The scheme $\mathbf{R} = \{R_1(ABC), R_2(AB)\}$ is not independent wrt $F = \{A \rightarrow BC\}$ but is ctm. As we will see in the Example 5, there is an algebraic-maintainable database scheme which is not ctm. In summary, independence implies constant-time-maintainability and constant-time-maintainability implies algebraic-maintainability. Moreover, the inclusions are proper.

3. Key-equivalent Database Schemes

In this section, we define a class of cover embedding database schemes which is neither independent nor γ-acyclic. The class of schemes properly contains a class of ctm schemes. We prove the desirabilities of this class of schemes by showing that it is bounded and algebraic-maintainable. We also characterize efficiently when such a scheme is ctm. Let $\mathbf{S}$ be a database scheme and F is its set of embedded key dependencies. Then $\mathbf{S}$ is said to be *key-equivalent* wrt F if for all S_i in $\mathbf{S}$, $S_i^+ = \cup\mathbf{S}$.

Example 3: Let $\mathbf{R}=\{R_1(AB), R_2(BC), R_3(AC)\}$, $F= \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B, C \rightarrow A, A \rightarrow C\}$. $\mathbf{R}$ is key-equivalent but $\mathbf{R}$ is not independent nor is γ-acyclic. In fact, $\mathbf{R}$ is not even α-acyclic [F3]. □

We want to prove first that key-equivalent database schemes are BCNF.

Lemma 3.1: Let $\mathbf{S}$ be a key-equivalent database scheme wrt the set of embedded key dependencies F. Then $\mathbf{S}$ is BCNF wrt F.

[Proof]: Suppose $\mathbf{S}$ is not BCNF wrt F. Then there is $X \rightarrow A \in F^+$ embedded in some $S \in \mathbf{S}$ such that X is not a superkey of S. Now observe that in this case, X must include a key in $\mathbf{S}$ because the set of fd's is a set of key dependencies. But since $\mathbf{S}$ is key-equivalent, $X \rightarrow S$ and hence X is a superkey of S. □

The above lemma guarantees that during the chasing of a state tableau of any state on a key-equivalent database, it is sufficient to equate symbols in whole tuples in the state tableau.

3.1. Key-equivalent Database Schemes are Bounded

We first show that any key-equivalent database scheme $\mathbf{S}$ is bounded. Let $\mathbf{s}$ be a consistent state on a key-equivalent database scheme $\mathbf{S}$. Algorithm 1, shown below, chases T_s, the state tableau for $\mathbf{s}$, to obtain the representative instance of $\mathbf{s}$. Notice that any key in a key-equivalent database scheme $\mathbf{S}$ functionally determines every attribute in $\cup\mathbf{S}$ and that step (1) of Algorithm 1 is well-defined since $\mathbf{s}$ is a consistent state.

Lemma 3.2: Let $\mathbf{s}$ be a consistent state on a database scheme $\mathbf{S}$ which is key-equivalent wrt F, where F is a set of key dependencies in $\mathbf{S}$. Let $\mathbf{K}$ be the set of keys embedded in $\mathbf{S}$. Let T_s and $\mathbf{K}$ be the input to Algorithm 1, and let T^* be the tableau returned after step (1) (but before step (2) is executed). Then

(a) if t is a tuple in T^* with its constant components defined on C, then $t[C]$ is a total tuple in $CHASE_F(T_s)$;

Algorithm 1

Input: The state tableau T_s for a consistent state **s** on a key-equivalent database scheme **S**, and a set of keys **K** in **S**.

Output: The representative instance of **s**.

Method:

(1) **while** (there are two tuples u and v in T_s that agree on a key K in **K** but their constant components are defined on distinct sets of attributes) **do**

Case (1): Whenever $v[A]$ is a constant, $u[A]$ is also a constant. In this case, equate the components of v to the corresponding constant components of u;

Case (2): The constant components of u and v are incomparable. In this case, $v[A]$ is equated to $u[A]$, wherever $u[A]$ is a constant;

end

(2) eliminate duplicate tuples with identical constant components from the tableau produced in step (1).

(b) if t is a tuple in T^* with its constant components defined on C, then $t[C]$ is returned by a join of a lossless subset of **S** covering C;

(c) any two tuples in T^* that agree on a key agree on their constant components.

[Proof]: (a) Step (1) of Algorithm 1 is an application of an fd-rule to tuples u and v involving the fd $K \rightarrow Y \in F^+$, for some $Y \subseteq U$. Hence the claim follows.

(b) We prove this by induction on the number k of fd-rules applied to T_s.

Basis: k=0. This is trivially true, because initially every total tuple in T_s originates from some relation scheme S_i.

Induction: k>0. Suppose the inductive hypothesis holds for $k-1$ applications of fd-rules. Now consider the k^{th} application of an fd-rule, or equivalently, the k^{th} iteration of the **while** loop. Let u and v be the two tuples involved. There are two cases to be considered, as are indicated in step (1) of the algorithm.

Case (1): Assume the components of v are set to the constant components of u. By the inductive hypothesis, the constant components of u can be computed by a join of a lossless subset of **S**. So after v is set to the constant components of u, the hypothesis still holds.

Case (2): By the inductive hypothesis, let E_u and E_v be the two joins of lossless subsets of **R** that compute the constant components of u and v respectively. After the corresponding components of v are set to u's, the constant components of v can be computed by $E_u \bowtie E_v$ which is a join of a lossless subset of **S** covering the constant components of v.

This completes the induction proof.

(c) If they would disagree on their constant components, we would not have finished with step (1). □

Corollary 3.1: Let **s** be a consistent state on a database scheme **S** which is key-equivalent wrt F, where F is a set of key dependencies in **S**. Let **K** be the set of keys embedded in **S**. Let T_s and **K** be the input to Algorithm 1 and let T_s^* be the final tableau produced by Algorithm 1. Then

(a) T_s^* is the representative instance for the state **s**, and all ndv's in T_s^* are distinct;

(b) the X-total projection of the representative instance is computed exactly by a union of projections onto X of all joins of lossless subsets of **R** covering X;

(c) **S** is bounded wrt F.

[Proof]: (a) The final tableau T_s^* is a satisfying relation wrt F. Hence it is the representative instance for the state **s**. Clearly all ndv's are distinct since no ndv's are being equated in the algorithm.

(b) Lemma 3.2(b) implies any X-total tuple is computed by some projection onto X of a join of a lossless subset of **R** covering X. By a result in [MUV], any projection onto X of a join of a lossless subset of **R** produces X-total tuples in the representative instance.

(c) Follows directly from (b) above. □

Example 4: Let $\mathbf{R}=\{R_1(AB), R_2(AC), R_3(AE), R_4(EB), R_5(EC), R_6(BCD), R_7(DA)\}$, $F = \{A\rightarrow B, A\rightarrow C, A\rightarrow E, E\rightarrow A, E\rightarrow B, E\rightarrow C, BC\rightarrow D, D\rightarrow BC, D\rightarrow A, A\rightarrow D\}$. **R** is key-equivalent wrt F. By Corollary 3.1(b), $[AE]$ is computed by $R_3 \cup \pi_{AE}(AB \bowtie AC \bowtie (BE \bowtie CE))$. Observe that the join expression is a union of projections of extension joins. □

3.2. Key-equivalent Database Schemes are Algebraic-maintainable

The last subsection shows that key-equivalent database schemes are bounded. In this subsection, we prove that they are algebraic-maintainable, but not necessarily ctm. Example 4 above can be used to show that key-equivalent database schemes are not ctm.

Example 5: Let $\mathbf{R}=\{R_1(AB), R_2(AC), R_3(AE), R_4(EB), R_5(EC), R_6(BCD), R_7(DA)\}$, $F = \{A\rightarrow B, A\rightarrow C, A\rightarrow E, E\rightarrow A\ E\rightarrow B, E\rightarrow C, BC\rightarrow D, D\rightarrow BC, A\rightarrow D, D\rightarrow A\}$. **R** is key-equivalent wrt F. Let us consider the following consistent state defined on the above database scheme.

$r_1(A\ B)$		$r_2(A\ C)$		$r_3(A\ E)$		$r_4(E\ B)$		$r_5(E\ C)$	
a	b	a	c			e_1	b	e_1	c
						e_2	b		
						.	.		
						.	.		
						.	.		
						e_n	b		

Consider now we insert the tuple $<a, e>$ into r_3. From the last example, at least we need to verify if $\{<a, e>\} \cup \pi_{AE}(AB \Join AC \Join (BE \Join CE))$ is satisfying wrt $A \rightarrow E$. Suppose **R** is ctm. Since the value "e" does not appear in r_4 or r_5, the way to verify if the updated state is consistent or not is by first issuing commands $\sigma_{A='a'}(R_1)$ and/or $\sigma_{A='a'}(R_2)$. Since $CST(\{<a,e>\} \bigcup \{\sigma_{A='a'}(R_1)\} \bigcup \{\sigma_{A='a'}(R_2)\}) = \{a,b,c,e\}$, we then can issue either $\sigma_{B='b'}(R_4)$ and/or $\sigma_{C='c'}(R_5)$. Without loss of generality, we assume $\sigma_{B='b'}(R_4)$ is issued. Then clearly given the above state, the number of tuples retrieved depends on the size of the state. If $\sigma_{C='c'}(R_5)$ is issued instead, a similar state can be constructed to show that the number of tuples retrieved depends on the state size. Hence **R** cannot be ctm wrt F. □

The above example shows that key-equivalent database schemes in general are not ctm. However, as we will see in the following, there is a simple algorithm to enforce satisfaction of constraints incrementally for key-equivalent database schemes via predetermined single-tuple conjunctive selections on some relational expressions. This will show that key-equivalent database schemes are algebraic-maintainable.

Consider now we are given a consistent state on a key-equivalent database scheme **S**. Suppose a tuple on $S_i \in \mathbf{S}$ is inserted into the consistent state. Algorithm 2, shown below, is an algorithm to verify if the resulting state from such an insertion is still consistent wrt the key dependencies in **S**. The following example illustrates how the algorithm works.

Example 6: Let $\mathbf{R} = \{R_1(ABE), R_2(AC), R_3(AD), R_4(BC), R_5(BD), R_6(CDE)\}$ and $F = \{A \rightarrow BE, B \rightarrow AE, E \rightarrow AB, A \rightarrow CD, B \rightarrow CD, E \rightarrow CD, CD \rightarrow E\}$. The set of keys in **R** is $\{A, B, E, CD\}$. **R** is key-equivalent wrt F. Let us consider the following state tableau for a consistent state on **R**. The *TAG*-column corresponds to the relation scheme from which the tuple originates. Since no fd-rule is applicable, the state tableau is also the representative instance for the state.

A	B	C	D	E	*TAG*
a		c			R_2
	b		d		R_5
		c	d	e	R_6

Suppose now we insert $<a, b, e'>$ into r_1. Since A, B, and E are the three keys in R_1, let the sequence of keys selected and processed by the first three iterations of **while** loop of Algorithm 2 be A, B and E. The three total tuples generated in step (4) for the keys A, B and E are $<a, c>$, $<b, d>$ and $<e'>$ respectively. So after the first three iterations of the **while** loop, $q = <a, b, c, d, e'>$.

At the beginning of the fourth iteration of the **while** loop, *unprocessed* $= \{CD\}$. Hence the total tuple v generated in step (4) in the fourth iteration of the **while** loop is $<c, d, e>$. But since in step (5), $q = <a, b, c, d, e'> \Join <c, d, e> = \emptyset$, the algorithm outputs *no*. The reader should verify that the updated state is not consistent wrt F. □

Theorem 3.1: Let a consistent state **s** on a key-equivalent database scheme **S**, a set of keys in **S**, and a tuple t on some $S_i \in \mathbf{S}$ be the input to Algorithm 2. The algorithm outputs *yes* exactly when $\mathbf{s} \cup \{t\}$ is consistent wrt the key dependencies in **S**.

Algorithm 2

Input: A consistent state **s** on a key-equivalent database scheme **S**, a set of keys in **S** and an inserted tuple t on some $S_i \in \mathbf{S}$.

Output: *yes* if $\mathbf{s} \cup \{t\}$ is consistent; *no* otherwise. If *yes* is printed, a tuple q is also output.

Method:

(1) Let $\{K_1, \ldots, K_u\}$ be the set of keys on S_i and
let $CHASE_F(T_s)$ be the representative instance for **s**;
unprocessed:=$\{K_1, \ldots, K_u\}$;
closure := S_i;
processed := $\emptyset$;
$q := t$;

(2) **while** (*unprocessed* $\neq \emptyset$) **do**

(3) let $K \in$ *unprocessed*;

(4) **if** there is a tuple p in $CHASE_F(T_s)$ such that $p[K] = q[K]$;
then $v := p[C]$; where C is the set of attributes on which p is constant;
else $v := q[K]$ and $C := K$;

(5) $q := q \Vert\!\bowtie\!\Vert v$;

(6) **if** q is empty, **then output** *no* and **exit**;

(7) *closure* := *closure*$\cup C$;

(8) let *new_keys* be the set of keys embedded in *closure* and
let *processed* := *processed*$\cup\{K\}$;

(9) *unprocessed* := *new_keys*$-$*processed*;

(10) **end**

(11) **output** *yes* and q.

[Proof]: Suppose the algorithm outputs *yes*. Let t be the tuple inserted into some $s_i \in \mathbf{s}$. Let t' be the tuple q produced by Algorithm 2 padded with distinct ndv's to $\cup\mathbf{S}$. We want to show that $CHASE_F(T_s)\cup\{t'\}$ is consistent. If t' does not agree with any tuple in $CHASE_F(T_s)$ on a key in **S**, then the final chased tableau for the state $\mathbf{s}\cup\{t\}$ is $CHASE_F(T_s)\cup\{t'\}$. Suppose one or more tuples agree with t' on a key. Let $\{p_1, \ldots, p_y\}$ be the set of tuples in $CHASE_F(T_s)$ that agree with t' on a key. By Algorithm 1, if p_m agrees with t' on a key K, there is no other tuple in $CHASE_F(T_s)$ that agree with t' on K. Since K is a key embedded in q, K will eventually be selected by statement (3) of Algorithm 2. By statements (4) and (5) of Algorithm 2, if $p_m[A]$ is a constant, then $t'[A]=p_m[A]$, for all $1\leq m\leq y$. So after the ndv's of the p_m's are equated to the corresponding constants from t', the set of tuples $\{p_1, \ldots, p_y, t'\}$ are identical on their constant components. Since no tuple other than $\{p_1, \ldots, p_y\}$ agrees

with t' on a key, no contradiction is possible in chasing $CHASE_F(T_s) \cup \{t'\}$ if the algorithm outputs *yes*. Thus the state $\mathbf{s} \cup \{t\}$ is consistent.

Now assume the algorithm outputs *no*. Suppose the rejection comes at the w^{th} iteration of the **while** loop. Let the sequence of keys processed up to the point of rejection be $K_1, \ldots, K_w$ and let $t_1', \ldots, t_w'$ be the sequence of total tuples generated in step (4) to extend t. Let $z = ((\cdots(t \bowtie t_1') \bowtie \cdots) \bowtie t_{w-1}')$. Clearly z is a total tuple in the chase of the state tableau for $\mathbf{s} \cup \{t\}$. By assumption, z is nonempty but $((\cdots(t \bowtie t_1') \bowtie \cdots) \bowtie t_w')$ is empty. Since $((\cdots(t \bowtie t_1') \bowtie \cdots) \bowtie t_w')$ is empty, $t_w'[A]$ and $z[A]$ are two distinct constants, for some $A \in (closure_w - K_w)$, where $closure_w$ is the *closure* at the beginning of the w^{th} iteration of the **while** loop. But $z[K_w] = t_w'[K_w]$. Hence $\mathbf{s} \cup \{t\}$ is inconsistent wrt the key dependencies. Therefore if the algorithm outputs *no*, then the updated state $\mathbf{s} \cup \{t\}$ is not consistent wrt F. □

Suppose we are given a consistent state $\mathbf{r}$ on a key-equivalent database scheme $\mathbf{R}$ and a key value $t[K]$. If we could compute the (unique) total tuple s, if it exists, in the representative instance for state $\mathbf{r}$ such that $t[K] = s[K]$ by applying to $\mathbf{r}$ a sequence of single-tuple conjunctive selections of the form $\sigma_\Phi(E)$ such that E and the size of the sequence depend on $\mathbf{R}$ and F, then together with Algorithm 2, $\mathbf{R}$ is algebraic-maintainable wrt its embedded key dependencies.

Let E_1 and E_2 be two lossless expressions of $\mathbf{R}$ on S_1 and S_2 respectively. Then E_1 is *greater than* E_2 if $S_1 \supset S_2$. Suppose K is a key in a key-equivalent database scheme $\mathbf{R}$. Let $\{E_1, \ldots, E_m\}$ be the set of joins of lossless subsets of $\mathbf{R}$ covering K. Since there is at least one relation scheme in $\mathbf{R}$ containing K, $m \geq 1$. Let $\mathbf{r}$ be a consistent state on $\mathbf{R}$. Then for any $1 \leq i \leq m$, $\sigma_{K = k}(E_i)$ will either return an empty set or a set containing a single tuple, or else the state is inconsistent. Hence $\sigma_{K = k}(E_i)$ is a single-tuple conjunctive selection. Let $\sigma_{K = k}(E_{i_1}), \ldots, \sigma_{K = k}(E_{i_p})$ be the expressions that return nonempty sets of tuples and let these sets of tuples be $\{t_1\}, \ldots, \{t_p\}$ respectively, $1 \leq p \leq m$. If $p \geq 1$, then there exists a $\{t_j\}$ produced by $\sigma_{K = k}(E_{i_j})$ such that $\sigma_{K = k}(E_{i_j})$ is greater than all other lossless expressions $\sigma_{K = k}(E_{i_q})$, $1 \leq q \neq j \leq p$. This is because $\sigma_{K = k}(E_{i_1} \bowtie \cdots \bowtie E_{i_p})$ is equivalent to $\sigma_{K = k}(E_{i_q})$, for some $1 \leq q \leq p$. By Lemma 3.2(c), Corollary 3.1(b) and the fact that $\sigma_{K = k}(E_{i_j})$ is greater than all other lossless expressions, t_j is the (unique) total tuple in the representative instance that contains the value "k". Since every total tuple with a particular key value can be found by means of a sequence of single-tuple conjunctive selections and its size depends only on $\mathbf{R}$ and F, together with Algorithm 2, any key-equivalent database scheme $\mathbf{R}$ is algebraic-maintainable wrt its embedded key dependencies.

Theorem 3.2: Let $\mathbf{R}$ be a key-equivalent database scheme wrt F, where F is a set of key dependencies embedded in $\mathbf{R}$. Then $\mathbf{R}$ is algebraic-maintainable wrt F.

[Proof]: Follows from the above argument. □

The following example illustrates how constraint enforcement can be carried out incrementally via relational expressions for key-equivalent database schemes.

Example 7: Let $\mathbf{R} = \{R_1(AB), R_2(AC), R_3(AE), R_4(EB), R_5(EC), R_6(BCD), R_7(DA)\}$ and $F = \{A \rightarrow B, A \rightarrow C, A \rightarrow E, E \rightarrow A, E \rightarrow B, E \rightarrow C, BC \rightarrow D, D \rightarrow BC, D \rightarrow A, A \rightarrow D\}$. Let us consider the following consistent state defined on the above database scheme.

$r_1(A$	$B)$
a	b

$r_2(A$	$C)$
a	c

$r_3(A$	$E)$

$r_4(E$	$B)$
e_1	b
e_2	b
.	.
.	.
e_n	b

$r_5(E$	$C)$
e_1	c

Consider now we insert the tuple $<a, e>$ into r_3. Since A and E are the two keys in R_3, by Algorithm 2, we have to compute the total tuples in the representative instance for the consistent state that embed the values "a" and "e" respectively. Suppose A is selected in the first iteration of the **while** loop in Algorithm 2. With the above state, the set of lossless subsets of **R** covering A which return a nonempty set of tuples is $\{R_1, R_2, R_1 \bowtie R_2, R_1 \bowtie R_2 \bowtie (R_4 \bowtie R_5)\}$. The total tuple in the representative instance that contains the value "a" is therefore computed by $\sigma_{A='a'}(R_1 \bowtie R_2 \bowtie (R_4 \bowtie R_5))$ and the tuple returned by this expression with the above consistent state is $<a, b, c, e_1>$. In step (5) of the algorithm, $<a, b, c, e_1> \bowtie <a, e> = \emptyset$. Hence Algorithm 2 outputs *no* and therefore the updated state is not consistent wrt its embedded key dependencies. □

3.3. Key-equivalent Database Schemes are Ctm if and only if Split-free

In the last subsection, we show that key-equivalent database schemes are algebraic-maintainable but not necessarily ctm. In this subsection we find a characterization of constant-time-maintainability for key-equivalent database schemes. Moreover the characterization can be tested efficiently under the key-equivalent assumption. We shall prove that key-equivalent database schemes are ctm exactly when their keys are not "split".

Let $\mathbf{S} = \{S_1, \ldots, S_m\}$ be a key-equivalent database scheme wrt $F = F_1 \cup \cdots \cup F_m$, where F_i is a set of key dependencies embedded in S_i, for all $1 \leq i \leq m$. We are going to define when **S** is split-free.

For any $S_l \in \mathbf{S}$, we compute S_l^+ wrt key dependencies embedded in **S** as shown below in Algorithm 3.

Let us consider a computation of S_l^+. Let $S_j \in \mathbf{S}$ be such that it is chosen in step (2) of Algorithm 3 and let $CP_j = S_j \cap closure'$, where $closure'$ is the *closure* when S_j is chosen in step (2). Then we say that S_j *completes a key* K *in* S_l^+ if $K \not\subseteq closure'$ and $S_j - CP_j \supseteq K - closure'$. Now we say that a key K is *split in* S_l^+ if there is a computation of S_l^+ where some S_j chosen in step (2) of Algorithm 3 that completes K in S_l^+ is such that $K \not\subseteq S_j$. Intuitively, K is split in S_l^+ if there is a partial computation of S_l^+ that covers K but none of the schemes in such a computation contains K. Then we say that S_l is *split-free (wrt F)* if no key in **S** is split in S_l^+; else it is *split*. **S** is *split-free* if for all $1 \leq i \leq m$, S_i is split-free; else it is *split*.

Example 8: Let $\mathbf{R} = \{R_1(AC), R_2(AB), R_3(BCA), R_4(BCD), R_5(AD)\}$ and $F = \{A \rightarrow C, A \rightarrow B, BC \rightarrow A, BC \rightarrow D, D \rightarrow BC, A \rightarrow BC, A \rightarrow D, D \rightarrow A\}$. **R** is split since the key BC is split in R_1^+, R_2^+, or R_5^+, but R_3 and R_4 are split-free. □

Algorithm 3

Input: $S_l \in \mathbf{S} = \{S_1, \ldots, S_m\}$ and $F = F_1 \cup \cdots \cup F_m$, where F_i is a set of key dependencies embedded in S_i, for all $1 \leq i \leq m$.

Output: S_l^+.

Method:

(1) $closure = S_l$;

(2) **while** (there is S_j in **S**, $S_j \not\subseteq closure$ and a key of S_j is included in $closure$) **do**

(3) $\quad closure = closure \cup S_j$;

(4) **end**

The following is an example of a split-free database scheme.

Example 9: Let $\mathbf{R} = \{R_1(AB), R_2(BC), R_3(CD), R_4(DE)\}$ and let $F = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B, C \rightarrow D, D \rightarrow C, D \rightarrow E, E \rightarrow D\}$. Since all keys consist of a single attribute, **R** is split-free. □

We claim that a key-equivalent database scheme is ctm if and only if it is split-free. We now start proving the *if*-part of this claim.

3.3.1. Key-equivalent Database Schemes are Ctm if Split-free

We assume for the rest of this subsection that $\mathbf{S} = \{S_1, \ldots, S_m\}$ is a split-free and key-equivalent database scheme wrt $F = F_1 \cup \cdots \cup F_m$, where F_i is a set of key dependencies embedded in S_i, for all $1 \leq i \leq m$. We want to prove that **S** is ctm. The plan for the proof is as follows. We first prove that $CHASE_F(T_s)$, for any consistent state **s** of **S**, can be computed using sequential extension joins. We then give an algorithm to solve the maintenance problem of **S** wrt F in constant time.

Let **s** be a consistent state on **S**. Algorithm 4, shown below, extends a tuple t on a key K embedded in **S** as far as possible using the key dependencies and tuples in **s**.

Lemma 3.3: Let **s** be a consistent state on a split-free key-equivalent database scheme **S**. Let $u[K]$, a tuple on some key K in **S**, and **s** be the input to Algorithm 4 and let t on C be the tuple returned by the algorithm. Then

(a) $t[C]$ is a total tuple in the final chased tableau for state **s**;

(b) if J is a key in **S** embedded in C and t' is the tuple on C' returned by Algorithm 4 with input $t[J]$ and **s**, then $C = C'$ and $t = t'$; and

(c) $t[C]$ can be computed by applying a sequence of single-tuple conjunctive selections on the instance $<\mathbf{s}, u[K]>$ to **s** and the size of the sequence is dependent on **S** and F. Moreover, the conjunctive selections are of the form $\sigma_\Phi(S_i)$, $S_i \in \mathbf{S}$.

Algorithm 4

Input: A tuple t on a key K embedded in **S**, and a consistent state **s** on **S**.

Output: A total tuple t' on C.

Method:

(1) $C = K$; $t'[K] = t[K]$ and $t'[A]$ is a distinct ndv, for all $A \in (\cup\ \mathbf{S}) - K$;

(2) **while** (there is a tuple p in some $s_i \in \mathbf{s}$ such that C includes a key K_i of S_i, $S_i - C \neq \emptyset$, and $p[K_i] = t'[K_i]$) **do**

(3) $\quad t'[S_i] = p[S_i]$; $C = C \cup S_i$;

(4) **end**

(5) **return** $t'[C]$ and C.

[Proof]: (a)(c) Follow trivially.

(b) Let C_k be the value of C after the k^{th} iteration of step (2) when $u[K]$ and **s** are used as input; we define $C_0 = K$. We prove by induction on k that for every key K' in C_k, $t'[K']$ can derive $t'[C_k]$ with input state **s**.

Basis: $k = 0$. It holds trivially.

Induction: $k > 0$. Suppose S_i is included into C_k in the k^{th} iteration of step (2) and suppose K' is a newly added key. From step (2), there is a key K_i of S_i such that $K_i \subseteq S_i \cap C_{k-1}$. By the inductive hypothesis, $t'[K_i]$ can derive $t'[C_{k-1}]$ with state **s**. Since S_i completes K' in C^+ and **S** is split-free, $K' \subseteq S_i$. By assumption that **S** is key-equivalent, $t'[K']$ can trivially derive $t'[S_i]$, and hence $t'[K_i]$, with state **s**. Thus $t'[K']$ can derive $t'[C_k]$ with state **s**.

This completes the inductive proof and hence $C = C'$ and $t'[C'] = t[C]$. □

Corollary 3.2: Let **s** be a consistent state on a split-free key-equivalent database scheme **S**. Let T^*_s be the final minimized chased tableau constructed for the state **s**. Then

(a) any X-total tuple in T^*_s can be obtained by a sequential extension join of some $S_i \in \mathbf{S}$ covering X; and

(b) there are no two tuples in T^*_s that agree on a key embedded in **S**, and all ndv's in T^*_s are distinct.

[Proof]: (a) Let T_s be the initial tableau for the state **s**. We obtain the final minimized chased tableau T^*_s in two steps as follows.

First step: For each relation $s_i \in \mathbf{s}$, and for each tuple $t \in s_i$, replace t by the tuple t' returned by Algorithm 4 with input $t[K]$ and **s**, where K is a key of S_i. Then pad t' to $\cup\mathbf{S}$ with distinct ndv's. Let T^* be the tableau produced in the first step. By Lemma 3.3(a), T^* is a partially chased tableau for **s**. By Lemma 3.3(b), for any two

tuples u and w in T^*, if u and w agree on a key, then they agree on their constant components.

Second step: We minimize T^* to obtain the final chased tableau. The minimization process involves eliminating tuples that agree on their constant components. After the elimination, it can be easily seen that the resulting tableau is the final chased tableau for T_s.

It follows that any X-total tuple in T^*_s can be obtained by a sequential extension join of some S_i covering X.

(b) It follows directly from (a). □

Given a consistent state on a split-free key-equivalent database scheme **S**, we want to show that there is a constant time algorithm to enforce satisfaction of fd's when a tuple is inserted into the state. Algorithm 5 shown below is such an algorithm.

Algorithm 5

Input: A consistent state **s** on a split-free key-equivalent database scheme **S**; a tuple t on S_i, where $S_i \in \mathbf{S}$.

Output: *yes* if $\mathbf{s} \cup \{t\}$ is consistent; otherwise, *no*.

Method:

(1) let $\{K_1, \ldots, K_u\}$ be the set of keys of S_i. For each K_j, invoke Algorithm 4 with input $t[K_j]$ and **s** and let t'_j on C_j be the tuple returned by Algorithm 4;

(2) let $q = \{t\} \bowtie \{t'_1\} \bowtie \cdots \bowtie \{t'_u\}$;

(3) **if** $q \neq \emptyset$, **then output** *yes* **else output** *no*.

Example 10: Let $\mathbf{S} = \{S_1(AB), S_2(BC), S_3(AC)\}$, and let $F = \{A \rightarrow B, B \rightarrow A, C \rightarrow B, B \rightarrow C, C \rightarrow A, A \rightarrow C\}$. The set of keys is $\{A, B, C\}$. **S** is split-free and key-equivalent. Let $\mathbf{s} = <s_1 = \{<a, b>\}, s_2 = \{<b, c>\}, s_3 = \emptyset>$. Suppose now we insert $<a, c'>$ into s_3. Let us execute Algorithm 5 with the above input. Since A and C are the two keys of S_3, $t'_1 = <a, b, c>$ and $t'_2 = <c'>$ are the two tuples returned in step (1). In step (2), $q = \{<a, c'>\} \bowtie \{<a, b, c>\} \bowtie \{<c'>\} = \emptyset$. Hence the algorithm outputs no. The reader can easily verify that the updated state is inconsistent wrt F. □

The following lemma proves that Algorithm 5 is correct. This shall imply that split-free key-equivalent database schemes are ctm.

Lemma 3.4: Let **s** be a consistent state on a split-free key-equivalent database scheme **S**. Let **s** and a tuple t on some $S_i \in \mathbf{S}$ be the input to Algorithm 5. The algorithm outputs *yes* exactly when $T^*_s \cup \{t'\}$ is consistent wrt the key dependencies embedded in **S**, where T^*_s is the final chased tableau for state **s**, and t' is the inserted tuple t on S_i padded with distinct ndv's to $\cup\mathbf{S}$.

[Proof]: Clearly if the algorithm outputs *no*, using an argument similar to the proof of Theorem 3.1, we can show that the updated state is inconsistent. If the algorithm outputs *yes*, we claim that $\{q'\} \cup T^*_s$ is consistent wrt F, where q' is the tuple q in Algorithm 5 padded to $\cup \mathbf{S}$ with distinct ndv's. Assume otherwise, there is a tuple μ in T^*_s that agree with q' on a key K, but is not one of t or t'_v's. By Corollary 3.2(b), K cannot be a key embedded completely in one of the relation schemes on which tuples t or t'_v's are defined. This implies K is split in S_i^+. A contradiction. Therefore we conclude that the resulting state with the inserted tuple is consistent. □

Now we are ready to state the main result in this subsection.

Theorem 3.3: Let $\mathbf{S} = \{S_1, \ldots, S_m\}$ be a database scheme key-equivalent wrt $F = F_1 \cup \cdots \cup F_m$, where F_i is a set of key dependencies embedded in S_i, for all $1 \leq i \leq m$. If $\mathbf{S}$ is split-free wrt F, then $\mathbf{S}$ is ctm wrt F.

[Proof]: From Lemma 3.4, Algorithm 5 is an algorithm that correctly determines if an updated state is consistent. Algorithm 5 invokes Algorithm 4 in time dependent on F only. By Lemma 3.3(c), each invocation is equivalent to applying a sequence of single-tuple conjunctive selections on the instance $<\mathbf{s}, t>$ to $\mathbf{s}$. Moreover, the size of the sequence is dependent on $\mathbf{S}$ and F. Hence $\mathbf{S}$ is ctm wrt F. □

3.3.2. Key-equivalent Ctm Database Schemes are Split-free

In this subsection, we prove that ctm key-equivalent database schemes are split-free. Let $\mathbf{S} = \{S_1, \ldots, S_m\}$ be key-equivalent wrt $F = F_1 \cup \cdots \cup F_m$, where F_i is a set of key dependencies embedded in S_i, for all $1 \leq i \leq m$. We shall prove that if $\mathbf{S}$ is split wrt F, then $\mathbf{S}$ is not ctm wrt F.

Assume $\mathbf{S}$ is split. Then there is a key K in $\mathbf{S}$ which is split in S_l^+, for some $S_l \in \mathbf{S}$. Let $S_{l_1}, \ldots, S_{l_k}$ be the relation schemes in a partial computation of S_l^+ that shows K is split, where $k > 1$. Let $\mathbf{S}_l = \{S_{l_1}, \ldots, S_{l_k}\}$ and let $U_l = \cup \mathbf{S}_l$. It is easy to observe that $U_l - K \neq \emptyset$. Let t_l be a tuple on U_l such that $t_l[B]$ is a unique constant, for every $B \in U_l$.

Now we construct a state $\mathbf{s}_l$ on $\mathbf{S}$ from t_l as follows:

- For all $S_i \in \mathbf{S}$, if $S_i \in \mathbf{S}_l$, then $s_i = \pi_{S_i}(\{t_l\})$, else $s_i = \emptyset$.

Now let $S_q \in \mathbf{S}$ be such that $S_q \supseteq K$; S_q does exist because K is a key of some scheme in $\mathbf{S}$; observe that $S_q \notin \mathbf{S}_l$ since no scheme in $\mathbf{S}_l$ may contain K. Let $S_{q_1}, \ldots, S_{q_{p+1}}$, where $p \geq 0$ and $S_{q_1} = S_q$, be a sequence of relation schemes in a partial computation of S_q^+ such that $S_{q_{p+1}} \cap (U_l - K) \neq \emptyset$, but $(S_{q_1} \cup \cdots \cup S_{q_p}) \cap (U_l - K) = \emptyset$; such computation exists since $S_q^+ = \cup \mathbf{S}$ and $U_l - K \neq \emptyset$. Let $\mathbf{S}_q = \{S_{q_1}, \ldots, S_{q_{p+1}}\}$ and let $U_q = \cup \mathbf{S}_q$.

Proposition 3.1: $\mathbf{S}_q \cap \mathbf{S}_l = \emptyset$.

[Proof]: If $p = 0$, then $S_{q_1} = S_q$ is the only member of $\mathbf{S}_q$. In this case, we already observed above that $S_q \notin \mathbf{S}_l$.

Assume $p > 0$. Suppose $S_{q_j} \notin \mathbf{S}_l$, for all $1 \leq j \leq p$. We claim $S_{q_{j+1}} \notin \mathbf{S}_l$. Assume otherwise. First observe that $(S_{q_1} \cup \cdots \cup S_{q_j}) \cap U_l = K$ and $(S_{q_1} \cup \cdots \cup S_{q_j}) \supseteq K'$, where K' is a key of $S_{q_{j+1}}$. These two facts imply $K \supseteq K'$, because $S_{q_{j+1}} \in \mathbf{S}_l$. But since no

scheme in $\mathbf{S}_l$ may contain K, $K \supset K'$. This violates the minimality condition for K to be a key, because $\mathbf{S}$ is key-equivalent. □

Now let $\mathbf{S}'_q = \emptyset$, if $p=0$, else let $\mathbf{S}'_q = \{S_{q_1}, \ldots, S_{q_p}\}$. Let $U'_q = \cup \mathbf{S}'_q$. Observe that $U_l \cap U'_q$ is either empty or is exactly K. However $U_l \cap U_q \supseteq AK$, $A \notin K$, and $K \rightarrow A \in F^+$. Let t_q be a tuple defined on U_q as follows: $t_q[K] = t_l[K]$, and $t_q[B]$ is a unique constant, for every $B \in U_q - K$; notice that $\{t_l[KA], t_q[KA]\}$ does not satisfy $K \rightarrow A$. Now we construct a state $\mathbf{s}'_q$ on $\mathbf{S}$ from $t_q[U'_q]$ as follows:

- For all $S_i \in \mathbf{S}$, if $S_i \in \mathbf{S}'_q$, then $s_i = \pi_{S_i}(\{t_q[U'_q]\})$, else $s_i = \emptyset$.

Let $\mathbf{s} = \mathbf{s}_l \cup \mathbf{s}'_q$.

Lemma 3.5: $\mathbf{s}$ is consistent wrt F.

[Proof]: We first chase tuples for states $\mathbf{s}_l$ and $\mathbf{s}'_q$ separately. Because of the losslessness of $\mathbf{S}_l$ and $\mathbf{S}'_q$, $t_l[U_l]$ and $t_q[U'_q]$ are derived. By considering the two cases that $U_l \cap U'_q = K$ or $\emptyset$, the lemma follows. □

Now we define a tuple u on $S_{q_{p+1}}$ such that $\mathbf{s} \cup \{u\}$ is inconsistent wrt F, where $u = t_q[S_{q_{p+1}}]$.

Lemma 3.6: $\mathbf{s} \cup \{u\}$ is inconsistent wrt F.

[Proof]: By definition, t_q and t_l violate $K \rightarrow A \in F^+$, where $KA \subseteq U_l \cap U_q$, $A \notin K$. Then the lemma follows since $\mathbf{S}_l$ and $\mathbf{S}_q$ are lossless wrt F. □

Now we are going to define some tuples such that $\mathbf{s}$ plus the new tuples are still consistent, but is inconsistent when the tuple u is included.

Lemma 3.7: Let $\{S_1, \ldots, S_r\} \subseteq \mathbf{S}_l$ be such that for $1 \leq h \leq r$, $K_h = S_h \cap K \neq \emptyset$. (Note that $r \geq 1$.) For $1 \leq h \leq r$, let $\mathbf{s}_h = \{t \mid t$ is a tuple on S_h such that $t[K_h] = t_l[K_h]$ and $t[B]$ is a unique constant for every $B \in S_h - K_h\}$, where $\mathbf{s}_h$ has an arbitrary but finite number of tuples. Then

(a) $\mathbf{s}_l \cup \mathbf{s}'_q \cup \mathbf{s}_1 \cup \cdots \cup \mathbf{s}_r$ is consistent wrt F;

(b) $\mathbf{s}'_q \cup \{u\} \cup \mathbf{s}_1 \cup \cdots \cup \mathbf{s}_r$ is consistent wrt F;

(c) $\mathbf{s}_l \cup \mathbf{s}'_q \cup \{u\} \cup \mathbf{s}_1 \cup \cdots \cup \mathbf{s}_r$ is inconsistent wrt F.

[Proof]: First observe that for $1 \leq h \leq r$, K_h is not a key in $\mathbf{S}$, or else the minimality condition of keys is violated.

For (a), from Lemma 3.5, $\mathbf{s} = \mathbf{s}_l \cup \mathbf{s}'_q$ is consistent wrt F. Now $\mathbf{s} \cup \mathbf{s}_1 \cup \cdots \cup \mathbf{s}_r$ is consistent wrt F because any two tuples in $CHASE_F(T_{\mathbf{s}})$ and $\mathbf{s}_1 \cup \cdots \cup \mathbf{s}_r$ are distinct on any key in $\mathbf{S}$.

For (b), $\mathbf{s}_q = \mathbf{s}'_q \cup \{u\}$ is consistent wrt F, since $\mathbf{s}_q$ is the projection of t_q on $\mathbf{S}_q$. Then $\mathbf{s}_q \cup \mathbf{s}_1 \cup \cdots \cup \mathbf{s}_r$ is consistent wrt F because any two tuples in $CHASE_F(T_{\mathbf{s}_q})$ and $\mathbf{s}_1 \cup \cdots \cup \mathbf{s}_r$ are distinct on any key in $\mathbf{S}$.

For (c), from Lemma 3.6 $\mathbf{s} \cup \{u\}$ is inconsistent wrt F, and so is $\mathbf{s} \cup \{u\} \cup \mathbf{s}_1 \cup \cdots \cup \mathbf{s}_r$. □

From parts (b) and (c) of Lemma 3.7, in order to show that $\mathbf{s}_l \cup \mathbf{s}'_q \cup \{u\} \cup \mathbf{s}_1 \cup \cdots \cup \mathbf{s}_r$ is inconsistent wrt F, we need tuples from $\mathbf{s}_l$ in addition to tuples in $\mathbf{s}'_q \cup \{u\} \cup \mathbf{s}_1 \cup \cdots \cup \mathbf{s}_r$.

We are ready now to prove our main claim in this subsection.

Theorem 3.4: Let $\mathbf{S} = \{S_1, \ldots, S_m\}$ be key-equivalent wrt $F = F_1 \cup \cdots \cup F_m$, where F_i is a set of key dependencies embedded in S_i, for all $1 \leq i \leq m$. Assume $\mathbf{S}$ is split wrt F. Then $\mathbf{S}$ is not ctm wrt F.

[Proof]: Assume $\mathbf{S}$ is ctm wrt F. Let us consider the state $\mathbf{s} = \mathbf{s}_l \cup \mathbf{s}'_q$ defined above. Then to each of the relations in $\mathbf{s}_l$ whose schemes contain a nonempty subset of K, let us add a distinct tuple as defined in Lemma 3.7; each of these relations now has two tuples; let $\mathbf{s}'_l$ be the resulting substate. Now let $\mathbf{s}' = \mathbf{s}'_l \cup \mathbf{s}'_q$. By Lemma 3.7(a), $\mathbf{s}'$ is consistent wrt F.

Let $\mathbf{s}'' = \mathbf{s}' \cup \{u\}$, where u is the tuple defined above on $S_{q_{p+1}}$. By Lemma 3.7(c), $\mathbf{s}''$ is inconsistent wrt F. Observe that u is a tuple on $S_{q_{p+1}}$, a scheme in $\mathbf{S}_q$, and then from Proposition 3.1, $u \notin \mathbf{s}_l$. Now Lemma 3.7(b) and 3.7(c) imply that tuples are needed from $\mathbf{s}_l$ to show $\mathbf{s}''$ is inconsistent wrt F.

Now observe that by construction of $\mathbf{s}''$, $\mathbf{s}_l$ and $\mathbf{s}''$—$\mathbf{s}_l$ have common attribute values only on tuples defined on schemes in $\mathbf{S}_l$ which contain nonempty subsets of K. The constant-time algorithm $\mathbf{A}$ will eventually try to access for the first time a tuple in $\mathbf{s}_l$ via a single-tuple conjunctive selection $\sigma_\Phi(\pi_X(S_v))$, where $S_v \in \mathbf{S}_l$. Before the request is issued, the values that are possibly known to $\mathbf{A}$ that are in the relation on S_v in state $\mathbf{s}_l$ are those in $\{t_q[B] \mid B \in K \cap S_v\}$, hence Φ is a formula of the form $K_v =$ 'k', where $K_v \subseteq K \cap S_v$. By a simple analysis, we could show that $X \supseteq K_v$. However, s_v, the relation on S_v, has two tuples with exactly the same values on K_v, hence the selection $\sigma_\Phi(\pi_X(S_v))$ is not single-tuple. Thus $\mathbf{A}$ does not solve the maintenance problem of $\mathbf{S}$ wrt F in constant time. □

Corollary 3.3: Let $\mathbf{S} = \{S_1, \ldots, S_m\}$ be key-equivalent wrt $F = F_1 \cup \cdots \cup F_m$, where F_i is a set of key dependencies embedded in S_i, for all $1 \leq i \leq m$. Then $\mathbf{S}$ is ctm iff $\mathbf{S}$ is split-free.

[Proof]: Follows from Theorems 3.3 and 3.4. □

3.3.3. An Efficient Test for Splitness

The following lemma proves that we can test splitness of key-equivalent database scheme in polynomial time.

Lemma 3.8: Let $\mathbf{R}$ be key-equivalent wrt its embedded key dependencies and let K be a key in $\mathbf{R}$. Let W be $\{R_p \mid R_p \in \mathbf{R}$ and R_p does not contain $K\}$ and let G be a set of key dependencies embedded in elements in W. The key K is split in R_i^+, for some $R_i \in \mathbf{R}$, iff there is a row t_j in $CHASE_G(T_W)$ such that $t_j[K]$ are all dv's.

[Proof]: Let $\mathbf{S}$ be any cover embedding database scheme wrt a set of fd's F. By a theorem in [BMSU], the chased tableau $CHASE_F(T_\mathbf{S})$ can be constructed as follows: For each row t_i in the tableau, say t_i corresponds to the relation scheme $S_i \in \mathbf{S}$, $t_i[A]$ is a dv if $A \in S_i^+$, where S_i^+ is computed wrt F, and distinct ndv's otherwise. We are now ready to prove our claim.

"If" Since W is cover embedding, the existence of such row t_j in $CHASE_G(T_W)$ implies there is a sequence of relation schemes from W that covers K. This shows that K is split in the closure of some relation scheme in W.

"Only if" If K is split in R_i^+, for some $R_i \in \mathbf{R}$, then there is a sequence of relation schemes that demonstrates K is split. This set of relation schemes in the sequence is a subset of W. Let t_i be the row in $CHASE_G(T_W)$ for R_i. If follows from the comment above that $t_i[K]$ are all dv's in $CHASE_G(T_W)$. □

4. Independence-reducible Database Schemes

In this section, we use the class of schemes characterized in the previous section to define a much larger class of schemes that has the same desirable properties.

Let $\mathbf{R}=\{R_1, \ldots, R_n\}$ be a database scheme and let $F = F_1 \cup \cdots \cup F_n$, where F_i is a set of key dependencies embedded in R_i, for all $1 \leq i \leq n$. $\mathbf{R}$ is said to be *key-equivalent independence-reducible wrt F*, or *independence-reducible wrt F* for short, if there is a partition $\mathbf{T}=\{T_1, \ldots, T_k\}$ of $\mathbf{R}$ such that

(a) $\mathbf{D} = \{\cup T_p \mid T_p \in \mathbf{T}\}$ is independent wrt F, and

(b) for any $T_p \in \mathbf{T}$, T_p is key-equivalent wrt the key dependencies embedded in elements in T_p.

If $\mathbf{R}$ is independence-reducible wrt F, then $\mathbf{T}$ is an *independence-reducible partition* of $\mathbf{R}$ and the database scheme $\mathbf{D}$ is the *corresponding independence-reducible database scheme* of $\mathbf{R}$ (induced by $\mathbf{T}$). $\mathbf{D}$ is obviously cover embedding wrt F and an embedded cover can be found by merging corresponding F_i's that are in the same block.

Example 11: Let $\mathbf{R}=\{R_1(AB), R_2(BC), R_3(AC), R_4(AD), R_5(DEF), R_6(DEG)\}$ and $F = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B, C \rightarrow A, A \rightarrow C, A \rightarrow D, D \rightarrow EFG\}$. $\mathbf{R}$ is independence-reducible. An independence-reducible partition is $\mathbf{T} = \{\{R_1, R_2, R_3, R_4\}, \{R_5, R_6\}\}$. Notice that $R_5^+ = R_6^+$. Similarly for R_1, R_2, R_3 and R_4. The corresponding independence-reducible database scheme in $\mathbf{D} = \{D_1(ABCD), D_2(DEFG)\}$ and is independent wrt F. □

Independent schemes have an interesting property that local satisfaction of constraints in each relation suffices to ensure global consistency of data. Independence-reducible schemes are a generalization of independent schemes in the sense that satisfaction of constraints within each block of relations in the partition guarantees global consistency of data. Consequently, the subset of relations that needs to be examined as a result of an insertion of a tuple can be readily identified. In the example above, given any state $\mathbf{r}$, if we can verify that the substates on $\{R_1,R_2,R_3,R_4\}$ and on $\{R_5,R_6\}$ are consistent wrt their respective embedded key dependencies, the state is guaranteed to be globally consistent. As we will show, independence-reducible schemes inherit most of the desirable properties of independent, as well as of key-equivalent, database schemes. We first prove that $\mathbf{D}$ is cover embedding BCNF wrt F.

Lemma 4.1: Let $\mathbf{R} = \{R_1, \ldots, R_n\}$ be cover embedding wrt a set of fd's F. Without loss of generality, let $F = F_1 \cup \cdots \cup F_n$, where F_i is a set of fd's embedded in R_i, for all $1 \leq i \leq n$. If there is some F_i that does not cover $F^+|R_i$, then $\mathbf{R}$ is not independent wrt F.

[Proof]: See [GY]. □

Corollary 4.1: Let **R** be independence-reducible wrt $F = F_1 \cup \cdots \cup F_n$, where F_i is a set of key dependencies embedded in R_i, $1 \leq i \leq n$. Let $\mathbf{T} = \{T_1, \ldots, T_k\}$ be an independence-reducible partition of **R**, let $\mathbf{D} = \{\cup T_p \mid T_p \in \mathbf{T}\}$ be the corresponding independence-reducible database scheme of **R**, and let F_p be the set of key dependencies embedded in elements in T_p, for all $1 \leq p \leq k$. Then **D** is cover embedding BCNF wrt F.

[Proof]: Clearly **D** is cover embedding. For BCNF wrt F: by Lemma 4.1, if $X \rightarrow A \in F^+$ is embedded in $\cup T_p$, then $X \rightarrow A \in F_p^+$. This implies X is a superkey of a relation scheme in T_p and by the assumption that T_p is key-equivalent, X is a superkey of $\cup T_p$. □

In the following subsections, we will prove some interesting properties about independence-reducible database schemes. These include that the class of independence-reducible database schemes is bounded and algebraic-maintainable wrt its embedded key dependencies. Sketches of algorithms are also given for computing total projections and for determining when an updated state is consistent.

4.1. Independence-reducible Database Schemes are Bounded

Let **r** be a consistent state on an independence-reducible database scheme **R**. Let $\mathbf{T} = \{T_1, \ldots, T_k\}$ be an independence-reducible partition of **R**. Let us construct a state **d** from **r** on the corresponding independence-reducible database scheme **D** as follows. For each $T_j \in \mathbf{T}$, let $T_j = \{S_1, \ldots, S_n\} \subseteq \mathbf{R}$ and $D_j = \cup T_j$. Let F_j be a set of key dependencies in T_j. Now construct a tableau on D_j from the substate $<s_1, \ldots, s_n>$ of **r** as follows: For each s_i on S_i in $<s_1, \ldots, s_n>$, pad s_i out to D_j with distinct ndv's, for all $1 \leq i \leq n$. Let the resulting tableau be T_{d_j}. Then chase T_{d_j} wrt F_j. Let the relation d_j be the final chased tableau. Note that d_j may contain some ndv's.

For each $D_j \in \mathbf{D}$, construct d_j as above and let the state $\mathbf{d} = <d_1, \ldots, d_k>$ be the *corresponding state* of **r**.

Lemma 4.2: Let **R** and **D** be as defined above. Then there is a sequence of fd-rules which converts T_r to a tableau which is equivalent to T_d, where **r** is a consistent state on **R**, **d** is its corresponding state on **D** constructed above, and T_d is the state tableau for **d**.

[Proof]: For each $D_j \in \mathbf{D}$, consider the part of the tableau in T_r for $S_1, \ldots, S_n$, where $D_j = \cup T_j = S_1 \cup \cdots \cup S_n$. We apply to T_r the same sequence of fd-rules as we applied to T_{d_j} in the construction of the relation d_j on D_j, for all $D_j \in \mathbf{D}$. Then T_d and the partially chased tableau for T_r are identical up to renaming of ndv's. □

Theorem 4.1: Let **R** be an independence-reducible database scheme wrt F, where F is a set of key dependencies embedded in **R**. Then **R** is bounded wrt F.

[Proof]: By Lemma 4.2, for any consistent state **r** on **R**, we can construct an equivalent state **d** on **D** such that the final chased tableaux are equivalent. By Corollary 4.1, **D** is cover embedding and BCNF wrt F. By the definition of independence-reducibility, **D** is independent wrt F.

Since **D** is cover embedding BCNF and independent wrt F, for any $X \subseteq U$, we can compute the X-total projection via an algebraic expression [S2]. The expression is a union of projections onto X of sequential extension joins covering X. Let $E(\mathbf{D}) = \pi_X(D_1 \bowtie \cdots \bowtie D_k)$ be an expression in such a union. Let $D_j = \cup T_j$, for any $1 \leq j \leq k$ and assume d_j is the final chased tableau for the substate of **r** on T_j wrt the key dependencies in T_j. For each D_j involved in $E(\mathbf{D})$, define $Y_j = D_j \cap (D_1 \cup \cdots \cup D_{j-1} \cup D_{j+1} \cup \cdots \cup D_k \cup X)$. Let $Y = \cup Y_i$. By Lemma 4.2 in [CA], $\pi_Y(\pi_{Y_1}(D_1) \bowtie \cdots \bowtie \pi_{Y_k}(D_k)) = \pi_Y(D_1 \bowtie \cdots \bowtie D_k)$. From definition, $X \subseteq Y$ and hence $\pi_X(\pi_{Y_1}(D_1) \bowtie \cdots \bowtie \pi_{Y_k}(D_k)) = E(\mathbf{D})$. Since we want the tuples returned by $E(\mathbf{D})$ to contain only constants in the state **r**, we claim $E(\mathbf{R}) = \pi_X(\lfloor Y_1 \rfloor \bowtie \cdots \bowtie \lfloor Y_k \rfloor)$ is an expression that returns exactly this set of tuples, where $\lfloor Y_j \rfloor$ is the set of Y_j-total tuples in d_j, for all $1 \leq j \leq k$.

If $k=1$, then $D_1 \supseteq X$ and hence $Y_1 = Y = X$. Then $E(\mathbf{R})$ clearly returns the correct answer. If $k>1$, and if any tuple $t \in \pi_{Y_j}(D_j)$ contains a ndv, then by assumption that T_j is key-equivalent and Corollary 3.1(a), the symbol is a distinct ndv. Let $t[A]$ be the entry in which the distinct ndv appears. Since $A \in Y_j$, either $A \in D_j \cap (D_1 \cup \cdots \cup D_{j-1} \cup D_{j+1} \cup \cdots \cup D_k)$ or $A \in D_j \cap X$. In the former case, t is not joinable with other tuples in the expression. In the latter case, even if t is joinable with other tuples to produce a tuple t' in $E(\mathbf{D})$, $t'[A]$ is a ndv and therefore t' is not a tuple we want to be included in $E(\mathbf{D})$. Hence $\pi_{Y_j}(D_j)$ cannot contain any ndv and our claim is proven. By assumption, T_j is cover embedding BCNF and key-equivalent wrt a set of key dependencies embedded in T_j, for all $1 \leq j \leq k$. By Corollary 3.1(b), the Y_j-total projection can be obtained by a union of joins of lossless subsets of T_j covering Y_j. Hence **R** is bounded wrt F. □

Example 12: Let $\mathbf{R} = \{R_1(AB), R_2(BC), R_3(AC), R_4(AD), R_5(DEF), R_6(DEG)\}$ and $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A, A \rightarrow D, D \rightarrow EFG\}$. **R** is independence-reducible. An independence-reducible partition **T** of **R** is $\{\{R_1, R_2, R_3, R_4\}, \{R_5, R_6\}\}$. The corresponding independence-reducible database scheme $\mathbf{D} = \{D_1(ABCD), D_2(DEFG)\}$ and is independent wrt F. Let us compute the ACG-total projection. The relational expression on **D** that computes the ACG-total projection is $\pi_{ACG}(D_1 \bowtie D_2)$. By the method in the proof of Theorem 4.1, we first find Y_1 and Y_2, where $Y_1 = D_1 \cap \{D_2 \cup ACG\} = ACD$ and $Y_2 = D_2 \cap \{D_1 \cup ACG\} = DG$. An expression to compute $\lfloor Y_1 \rfloor$ is $\pi_{ACD}(R_1 \bowtie R_2 \bowtie R_4) \cup \pi_{ACD}(R_3 \bowtie R_4)$. An expression to compute $\lfloor Y_2 \rfloor$ is $\pi_{DG}(R_6)$. So an expression to compute the ACG-total projection is $\pi_{ACG}(\lfloor Y_1 \rfloor \bowtie \lfloor Y_2 \rfloor) = \pi_{ACG}((\pi_{ACD}(R_1 \bowtie R_2 \bowtie R_4) \cup \pi_{ACD}(R_3 \bowtie R_4)) \bowtie \pi_{DG}(R_6))$. □

4.2. Independence-reducible Database Schemes are Algebraic-maintainable

Constraint enforcement for an independence-reducible database scheme **R** can be carried out incrementally. Suppose $r_m \in \mathbf{r}$ is updated by an insertion of a tuple. By definition of independence-reducibility, **R** is partitioned into blocks $T_1, \ldots, T_k$ and each T_j is key-equivalent wrt its key dependencies. Let $R_m \in T_j$, for some T_j. By Theorem 3.2, any key-equivalent database scheme is algebraic-maintainable wrt its embedded key dependencies. So after a tuple on R_m is inserted into a consistent state **r** on **R**, we could determine if the updated state on T_j is consistent wrt its embedded key dependencies. If the updated state on T_j is not consistent, then obviously the

updated state on **R** is not consistent. Suppose the updated state on T_j is consistent, we claim that the updated state is also consistent. Since **R** is partitioned by $\{T_1, \ldots, T_k\}$, relations in the updated state are also partitioned by $\{T_1, \ldots, T_k\}$. For all $T_i = \{S_1, \ldots, S_p\}$, $i \neq j$, the final chased tableau d_i for the substate $<s_1, \ldots, s_p>$ of the updated state on T_i is consistent, since by assumption the original state is consistent. By assumption that the updated substate on T_j is consistent, hence every chased state tableau d_l on T_l is consistent wrt its embedded key dependencies, for all $1 \leq l \leq k$. By definition of independence-reducibility, $\{\cup T_1, \ldots, \cup T_k\}$ is independent wrt F. By considering each chased state tableau as a relation, $\mathbf{d} = <d_1, \ldots, d_k>$ is a state on $\{\cup T_1, \ldots, \cup T_k\}$. Since each d_l is satisfying wrt its embedded fd's, the updated state is consistent wrt F. Hence our claim is proven.

Theorem 4.2: Let **R** be an independence-reducible database scheme wrt F, where F is a set of key dependencies embedded in **R**. Then **R** is algebraic-maintainable wrt F.

[Proof]: Follows from the above argument. □

4.3. More Properties of Independence-reducible Database Schemes

The following is another interesting property of the class of independence-reducible database schemes. Let **R** be a database scheme. $SUBSET(\mathbf{R}) = \{R_i \mid R_i$ is a nonempty subset of some R_j in $\mathbf{R}\}$. AUG(**R**) = **R** ∪ **S**, where $\mathbf{S} \subseteq SUBSET(\mathbf{R})$. It is worth noting that given a database scheme **R**, there may be many AUG(**R**)'s. Let **C** be a class of database schemes. Then AUG(**C**) = {AUG(**R**) | **R** is in **C**}. We call AUG(**C**) the *augmentation* of **C**. Let **C** be the class of independence-reducible database schemes. We want to show that this class of schemes is closed under the augmentation operation. That is, AUG(**C**) = **C**. First we need the following property of independent schemes.

Lemma 4.3: Let **R** be cover embedding BCNF and independent wrt F. Let $S \subseteq U$ and S does not embed any key of **R**. Then $\mathbf{R} \cup \{S\}$ is independent wrt F.

[Proof]: First observe that $\mathbf{R} \cup \{S\}$ is cover embedding BCNF wrt F. Suppose $\mathbf{R} \cup \{S\}$ is not independent wrt F. Then there are R_i and R_j in $\mathbf{R} \cup \{S\}$ such that R_i^+ wrt F-F_j contains a key dependency in F_j^+. Since F_j contains some nontrivial fd, $R_j \neq S$. Since $S^+ = S$ and S embeds only trivial fd's, $R_i \neq S$. This implies **R** is not independent wrt F. □

Theorem 4.3: Let **C** be the class of independence-reducible database schemes. Then AUG(**C**) = **C**.

[Proof]: By definition, AUG(**C**) contains **C**. Now we need to show every element in AUG(**C**) is also in **C**. We prove this by showing that if **R** is an independence-reducible database scheme, then $\mathbf{V} = \mathbf{R} \cup \{S\}$ is also independence-reducible, where S is a nonempty proper subset of some $R_j \in \mathbf{R}$. Let **T** be an independence-reducible partition of **R** and **D** its corresponding independence-reducible database scheme.

Case (1): S does not contain a key of any $R_j \in \mathbf{R}$. In this case, an independence-reducible partition for **V** is $\mathbf{T} \cup \{\{S\}\}$ and the corresponding database scheme for **V** is $\mathbf{D} \cup \{S\}$. Since **D** is independent wrt F and S embeds no key of any $R_j \in \mathbf{R}$, by Lemma 4.3, $\mathbf{D} \cup \{S\}$ is independent wrt F. Hence **V** is also independence-reducible

wrt F.

Case (2): S embeds some key of some R_j. In this case all the keys embedded in S are equivalent. Let these keys be keys of R_i, for some $R_i \in \mathbf{R}$. Assume further that R_i is in the block T_i in $\mathbf{T}$. Then an independence-reducible partition of $\mathbf{V}$ is $\mathbf{T}'$, where $\mathbf{T}'$ is $\mathbf{T}$ except that T_i includes S. The corresponding database scheme for $\mathbf{V}$ is still $\mathbf{D}$. Hence $\mathbf{V}$ is independence-reducible.

Therefore AUG($\mathbf{C}$) = $\mathbf{C}$. □

Let $\mathbf{R}$ be a database scheme. RED($\mathbf{R}$) is the *reduction* of $\mathbf{R}$. A database scheme $\mathbf{R}$ is *reduced* if no relation scheme is a proper subset of other.

Corollary 4.2: Let $\mathbf{R}$ be a database scheme and F be a set of embedded key dependencies. $\mathbf{R}$ is independence-reducible wrt F iff RED($\mathbf{R}$) is.

[Proof]: Follows from Theorem 4.3. □

5. Recognition and Subclasses of Independence-reducible Database Schemes

In this section, we shall derive an efficient algorithm that recognizes exactly the class of independence-reducible database schemes. We also state the condition under which an independence-reducible database scheme is ctm. We prove some interesting properties of the class of schemes accepted by the recognition algorithm. These results will show that this class of schemes properly contains a superset of the classes of schemes identified by Sagiv [S1][S2] and by Chan and Hernández [CH1]. This proves that the class of independence-reducible database schemes is the largest known class of schemes which is desirable wrt query answering and constraint enforcement when a set of key dependencies is considered.

5.1. Finding the Key-equivalent Partition of R

Let $\mathbf{R}$ be a cover embedding database scheme and let $R_i \in \mathbf{R}$. Define $[R_i]$ as the largest subset of $\mathbf{R}$ containing R_i such that $[R_i]$ is key-equivalent wrt its embedded key dependencies. The collection $\{[R_i] \mid R_i \in \mathbf{R}\}$ is called the *key-equivalent partition* of $\mathbf{R}$. The key-equivalent partition of $\mathbf{R}$ is unique.

Let $\mathbf{R} = \{R_1, \ldots, R_n\}$ be a database scheme and let $F = F_1 \cup \cdots \cup F_n$, where F_i is a set of key dependencies embedded in R_i, for all $1 \leq i \leq n$. We shall prove that the output from *KEP*, shown below, when ($\mathbf{R}$, F) is its input, is the key-equivalent partition of $\mathbf{R}$.

It is easy to see that the output of *KEP*($\mathbf{R}$, F) is a partition of $\mathbf{R}$. The following example illustrates how the key-equivalent partition is obtained by *KEP*.

Example 13: Let $\mathbf{R} = \{R_1(AB), R_2(CD), R_3(ABC), R_4(ABD), R_5(CDE), R_6(EA), R_7(EF), R_8(FB)\}$ and let $F = \{AB \rightarrow C, AB \rightarrow D, CD \rightarrow E, E \rightarrow CD, E \rightarrow A, E \rightarrow F, F \rightarrow B\}$.

Assume we call *KEP* with $\mathbf{R}$ and F as its input. Since $R_1^+ = \cdots = R_7^+ = ABCDEF$ and $R_8^+ = FB$, then *part* in statement (2) is $\{[R_1] = \{R_1, \ldots, R_7\}, [R_8] = \{R_8\}\}$. Thus the union of *KEP*($\{R_1, \ldots, R_7\}$, G_1) and *KEP*($\{R_8\}$, G_2) is the key-equivalent partition of $\mathbf{R}$, where G_1 and G_2 are a set of key dependencies embedded in elements of $[R_1]$ and $[R_8]$ respectively.

---

function *KEP*(**R**, F);

Input: A database scheme $\mathbf{R} = \{R_1, \ldots, R_n\}$ and $F = F_1 \cup \cdots \cup F_n$, where F_i is a set of key dependencies embedded in R_i, for all $1 \leq i \leq n$.

Output: The key-equivalent partition of **R** wrt F.

Notation: $[R_j] = \{R_i \in \mathbf{R} \mid R_i^+ = R_j^+\}$.

Method:

(1) **begin**

(2) let $part = \{ [R_j] \mid R_j \in \mathbf{R}\}$;

(3) **if** $part = \{ \mathbf{R} \}$ **then return**($\{\mathbf{R}\}$)
else return($KEP(p_1, G_1) \cup \cdots \cup KEP(p_l, G_l)$), where $part = \{p_1, \ldots, p_l\}$ and G_j, for all $1 \leq j \leq l$, is a set of key dependencies embedded in schemes of p_j;

(4) **end**

For $KEP(\{R_1, \ldots, R_7\}, G_1)$, *part* in statement (2) is $\{ [R_1] = \{R_1, R_3, R_4\}, [R_2] = \{R_2, R_5, R_6, R_7\}\}$ (remember that closures are computed wrt $G_1 = F_1 \cup \cdots \cup F_7$). Thus we have to compute the key-equivalent partitions of $\{R_1, R_3, R_4\}$ and $\{R_2, R_5, R_6, R_7\}$ wrt key dependencies embedded in $\{R_1, R_3, R_4\}$ and $\{R_2, R_5, R_6, R_7\}$ respectively. The sets returned contain the sets input. Hence $KEP(\{R_1, \ldots, R_7\}, G_1)$ returns $\{\{R_1, R_3, R_4\}, \{R_2, R_5, R_6, R_7\}\}$.

Since $KEP(\{R_8\}, G_2) = \{\{R_8\}\}$, the key-equivalent partition of **R** is $\{\{R_8\}, \{R_1, R_3, R_4\}, \{R_2, R_5, R_6, R_7\}\}$. □

The following is a basic fact about *KEP* and its correctness follows from statements (2) and (3) of the function.

Lemma 5.1: Let $\mathbf{R} = \{R_1, \ldots, R_n\}$ be a database scheme and let $F = F_1 \cup \cdots \cup F_n$, where F_i is a set of key dependencies embedded in R_i, for all $1 \leq i \leq n$. Let $\{KE_1, \ldots, KE_l\}$ be the output from *KEP* when (**R**, F) is its input. Then for all $1 \leq i \leq l$, KE_i is key-equivalent wrt its embedded key dependencies.

Lemma 5.2: Let $\mathbf{R} = \{R_1, \ldots, R_n\}$ be a database scheme and let $F = F_1 \cup \cdots \cup F_n$, where F_i is a set of key dependencies embedded in R_i, for all $1 \leq i \leq n$. Let $\{KE_1, \ldots, KE_l\}$ be the output from *KEP* when (**R**, F) is its input. Assume $\mathbf{S} \subseteq \mathbf{R}$ is key-equivalent wrt its embedded key dependencies. Then $\mathbf{S} \subseteq KE_i$, for some $1 \leq i \leq l$.

[Proof]: This follows from the fact that if $\mathbf{S} \subseteq \mathbf{R}$ and **S** is key-equivalent wrt its embedded key dependencies, then in statement (3), either $part = \{\mathbf{R}\}$, in which case the lemma holds, or in any recursive invocation of the function, $\mathbf{S} \subseteq p_q$, for some p_q in

statement (3) of the function. □

Lemmas 5.1 and 5.2 prove that the partition of **R** produced by *KEP* is the key-equivalent partition of **R**.

5.2. Recognition of Independence-reducible Database Schemes

In this subsection we give an efficient algorithm that recognizes exactly the class of independence-reducible database schemes.

Algorithm 6

Input: A database scheme **R** and a set G of key dependencies embedded in **R**.

Output: *Accept* or *reject*. If *accept* is output, an independence-reducible partition of **R** and an embedded cover are also output.

Method:

(1) generate the key-equivalent partition $\{KE_1, \ldots, KE_n\}$ of **R** via $KEP(\mathbf{R}, G)$;

(2) let F_j be a set of key dependencies embedded in elements in KE_j, for all $1 \leq j \leq n$;

(3) **if** $\{\cup KE_1, \ldots, \cup KE_n\}$ is not independent wrt G (or $\cup F_j$), **then** output *reject*, **else do** output *accept*, $\{F_1, \ldots, F_n\}$, and $\{KE_1, \ldots, KE_n\}$ **end.**

Suppose $\{KE_1, \ldots, KE_n\}$ is the key-equivalent partition of **R** produced after the execution of step (1) when $(\mathbf{R}, G)$ is used as input to Algorithm 6. Suppose $(\mathbf{R}, G)$ is accepted by Algorithm 6 and let $\mathbf{D} = \{\cup KE_1, \ldots, \cup KE_n\}$ be the database scheme and $\{F_1, \ldots, F_n\}$ be its corresponding embedded cover. Note that G is equivalent to $\bigcup_{i=1}^{n} F_i$.

Corollary 5.1: Let **R** be a database scheme and let G be a set of key dependencies embedded in **R**. If $(\mathbf{R}, G)$ is accepted by Algorithm 6, then **R** is independence-reducible wrt G.

[Proof]: Let $\mathbf{T} = \{KE_1, \ldots, KE_n\}$ be the partition of **R** generated in step (1) of Algorithm 6. By Lemma 5.1, KE_j is key-equivalent wrt its embedded key dependencies. By step (3) of Algorithm 6, $\mathbf{D} = \{\cup KE_j = D_j \mid KE_j \in \mathbf{T}\}$ is independent wrt G. Hence **R** is independence-reducible wrt G. □

Corollary 5.2: Let **R** be a database scheme and let G be the set of key dependencies embedded in **R**. If $(\mathbf{R}, G)$ is accepted by Algorithm 6, then **R** is bounded and algebraic-maintainable wrt G.

[Proof]: Follows directly from Theorems 4.1, 4.2 and Corollary 5.1. □

Corollary 5.3: Let **R** be an independence-reducible database scheme wrt G, where G is the set of key dependencies embedded in **R**. Let $\{P_1, \ldots, P_k\}$ be an independence-reducible partition of **R** and let $\{KE_1, \ldots, KE_n\}$ be the partition

generated after the execution of step (1) of Algorithm 6 when $(\mathbf{R}, G)$ is its input. Then for all $1 \leq i \leq k$, there is exactly one KE_q such that $P_i \subseteq KE_q$, for some $1 \leq q \leq n$.

[Proof]: By Lemma 5.2, for each P_i, $1 \leq i \leq k$, $P_i \subseteq KE_q$, for some $1 \leq q \leq n$. If P_i is embedded in more than one KE_q, then $\{KE_1, \ldots, KE_n\}$ does not partition $\mathbf{R}$. Hence there is exactly one KE_q that embeds P_i. □

Theorem 5.1: Let $\mathbf{R}$ be an independence-reducible database scheme wrt G, where G is the set of key dependencies embedded in $\mathbf{R}$. Then $(\mathbf{R}, G)$ is accepted by Algorithm 6.

[Proof]: Let $\{KE_1, \ldots, KE_n\}$ be the partition generated after the execution of step (1) when $(\mathbf{R}, G)$ is input to the algorithm. Let $\{P_1, \ldots, P_k\}$ be an independence-reducible partition of $\mathbf{R}$. Corollary 5.3 implies $\{P_1, \ldots, P_k\}$ is partitioned into n disjoint sets. Let $P_{q_0}, \ldots, P_{q_m}$ be the blocks contained in some KE_q, for some $1 \leq q \leq n$. It is easy to verify that $\bigcup_{i=0}^{m} P_{q_i} = KE_q$.

Let $\mathbf{D} = \{D_1 = \cup KE_1, \ldots, D_n = \cup KE_n\}$ and $F = F_1 \cup \cdots \cup F_n$ be the key dependencies embedded in $\mathbf{D}$. We want to show that if $\{\cup P_1, \ldots, \cup P_k\}$ is independent wrt its embedded key dependencies, then $\mathbf{D}$ is also independent wrt F.

Suppose $\mathbf{D}$ is not independent wrt F. By a close inspection of the algorithm for testing independence from [GY] and by Lemma 5.1 that KE_v is key-equivalent wrt its embedded key dependencies, for all $1 \leq v \leq n$, there are D_i, D_j, $i \neq j$, and there is a sequence of relation schemes $D_i = D_{i_1}, \ldots, D_{i_l}$ such that $D_{i_p} \neq D_j$, for all $1 \leq p \leq l$, $\bigcup_{p=1}^{l} D_{i_p} \cap D_j = X$ and X embeds a key dependency $K_j \rightarrow A$ in F_j^+.

Consider $\mathbf{S} \subseteq \mathbf{R}$ for those relation schemes in $\mathbf{R}$ that are members of D_{i_p}, $1 \leq p \leq l$. Clearly $\{D_{i_1}, \ldots, D_{i_l}\}$ is connected and lossless wrt its embedded key dependencies. KE_{i_p} is also connected and lossless wrt its embedded key dependencies, where $D_{i_p} = \cup KE_{i_p}$, for all $1 \leq p \leq l$. Hence $\mathbf{S}$ is lossless wrt its embedded key dependencies.

Let $\cup KE_j = D_j$. Since $K_j \rightarrow A$ is in F_j^+, there is a sequence of key dependencies, and hence a sequence of the relation schemes in KE_j of K_j covering A. Let the sequence be $H_1, \ldots, H_q$. Without loss of generality, we assume for all H_p, $p < q$, H_p does not contain A. Then $\mathbf{V} = \mathbf{S} \cup \{H_1, \ldots, H_{q-1}\}$ contain $K_q A$, for a key dependency $K_q \rightarrow A$ in H_q.

Let R_p be any relation scheme in KE_i. Let G_{H_q} be a set of key dependencies embedded in H_q. Then R_p^+ wrt G - G_{H_q} contains $\mathbf{V}$ and hence R_p and H_q violate the independence test since R_p^+ wrt G - G_{H_q} contains $K_q \rightarrow A$, which is a key dependency embedded in H_q. Hence $\mathbf{R}$ is not independent wrt its embedded key dependencies. Since R_p and H_q are not in the same KE_v, they are not in the same P_s. This implies that $\{\cup P_1, \ldots, \cup P_k\}$ is not independent wrt its embedded key dependencies. □

Corollary 5.4: Let $\mathbf{R}$ be a database scheme and F be the set of embedded key dependencies. Then there is a polynomial time algorithm that determines if $\mathbf{R}$ is independence-reducible wrt F.

[Proof]: Algorithm 6 clearly is a polynomial time algorithm and recognizes exactly the class of independence-reducible database schemes. Its correctness follows from Corollary 5.1 and Theorem 5.1. □

5.3. BCNF Independent and BCNF γ-acyclic Schemes are Independence-reducible

In this subsection, we will show that the class of database schemes recognized by Algorithm 6 properly includes all previously known classes of database schemes which are bounded wrt a set of embedded key dependencies.

Theorem 5.2: Let **R** be a cover embedding γ-acyclic BCNF database scheme wrt G. Then (**R**, G) is accepted by Algorithm 6.

[Proof]: Suppose there is a cover embedding γ-acyclic BCNF database scheme (**R**, G) that is rejected by Algorithm 6. Let $\mathbf{D} = \{D_1 = \cup KE_1, \ldots, D_n = \cup KE_n\}$ be the database scheme that is rejected by the independence test in step (3) of Algorithm 6. Let F_k be a set of key dependencies embedded in D_k, for all $1 \leq k \leq n$ and let $F = \cup F_k$.

Since **D** is not independent wrt F, by a close inspection of the algorithm for testing independence from [GY] and by Lemma 5.1 that KE_v is key-equivalent wrt its embedded key dependencies, for all $1 \leq v \leq n$, there are D_i, D_j, $i \neq j$, and there is a sequence of relation schemes $D_i = D_{i_1}, \ldots, D_{i_m}$ such that $D_{i_p} \neq D_j$, for all $1 \leq p \leq m$, $\cup D_{i_p} \cap D_j = X$ and X embeds a key dependency $K_j \rightarrow A$ in F_j^+.

Consider the hypergraph **S** for those relation schemes in **R** that are members of D_{i_p}, for all $1 \leq p \leq m$. Clearly $\{D_{i_1}, \ldots, D_{i_m}\}$ is connected. KE_{i_p} is also connected, where $D_{i_p} = \cup KE_{i_p}$, for all $1 \leq p \leq m$. Hence **S** is a connected subhypergraph of **R**.

Let $\cup KE_j = D_j$. Since $K_j \rightarrow A$ is in F_j^+, there is a sequence of key dependencies, and hence a sequence of the relation schemes in KE_j of K_j covering A. Let the sequence be $H_1, \ldots, H_q$. Without loss of generality, we assume for all H_p, $p < q$, H_p does not contain A. Then $\mathbf{V} = \mathbf{S} \cup \{H_1, \ldots, H_{q-1}\}$ contains K_qA, where $K_q \rightarrow A$ is a key dependency in H_q. If K_qA is embedded in some element in **S**, then H_q is in some D_{i_p}, for some $1 \leq p \leq m$. This would contradict $\{KE_1, \ldots, KE_n\}$ partitions **R**. By assumption A is not contained in any H_p, $p < q$, K_qA is not embedded in any element in **V**.

Since no element in **V** contains K_qA and **V** is a connected subhypergraph of **R**, a u.m.c. does not exist for K_qA. By Theorem 2.1, **R** is not γ-acyclic. □

Next we want to show that the class of database schemes recognized by the algorithm properly contains the class of independent schemes identified by Sagiv [S2].

Theorem 5.3: Let $\mathbf{R} = \{R_1, \ldots, R_v\}$ be a cover embedding independent database scheme wrt its embedded key dependencies $G = G_1 \cup \cdots \cup G_v$. Then (**R**, G) is accepted by Algorithm 6.

[Proof]: $\{\{R_1\}, \ldots, \{R_v\}\}$ is an independence-reducible partition and the theorem follows trivially. □

As a consequence of the above result, the class of database schemes accepted by Algorithm 6 in fact properly contains a superset of the previously known classes of bounded (which are also ctm) database schemes.

Theorem 5.4: AUG(**S**) and AUG(**G**) are both accepted by Algorithm 6 and hence are both bounded and algebraic-maintainable, where **S** is the class of cover embedding BCNF independent schemes while **G** is the class of γ-acyclic cover embedding BCNF database schemes.

[Proof]: Follows directly from Theorems 4.3, 5.2, 5.3 and Corollary 5.4. □

5.4. A Characterization of Ctm for Independence-reducible Schemes

We showed in Section 4 that independence-reducible schemes are algebraic-maintainable, but not necessarily ctm. We now state the condition under which an independence-reducible scheme is ctm. Let **R** be an independence-reducible database scheme wrt F, where F is a set of key dependencies embedded in **R**. Let $\mathbf{T} = \{T_1, \ldots, T_k\}$ be an independence-reducible partition of **R**. Then we say that **R** is *split-free* if each T_i in **T** is split-free wrt its embedded key dependencies.

Theorem 5.5: Let **R** be an independence-reducible database scheme wrt F, where F is a set of key dependencies embedded in **R**. **R** is ctm if and only if **R** is split-free.

[Proof]: It follows from definition of independence-reducibility and Corollary 3.3. □

6. Conclusion

We defined a generalization of independent schemes, called independence-reducible schemes, and proved that it is highly desirable with respect to query answering and constraint enforcement. The criteria we used in evaluating a database scheme are boundedness and algebraic-maintainability. We showed that this class of schemes is bounded by deriving relational expressions for computing total projections. We proved that it is algebraic-maintainable by finding an incremental algorithm for enforcing constraints via single-tuple conjunctive selections. To demonstrate that the class of schemes identified is quite general, we showed that it includes a superset of all previously known classes of cover embedding BCNF database schemes with similar desirable properties. We also found an efficient algorithm which recognizes exactly the class of independence-reducible database schemes. Independence-reducible schemes properly contain a class of ctm schemes. An efficient test was found which determines if an independence-reducible scheme is ctm.

References

[ABU] Aho, A.V., Beeri, C., Ullman, J.D., "The Theory of Joins in Relational Databases," *ACM TODS 4:3 (1979)*, pp. 297-314.

[ASU] Aho, A.V., Sagiv, Y., Ullman, J.D., "Equivalence of Relational Expressions," *SIAM J. on Computing 8:2 (1979)*, pp. 218-246.

[AC1] Atzeni, P., Chan, E.P.F., "Efficient Query Answering in the Representative Instance Approach," *Proc. ACM PODS 1985*, pp. 181-188.

[AC2] Atzeni, P., Chan, E.P.F., "Independent Database Schemes Under Functional and Inclusion Dependencies," *Proc. VLDB 1987*, pp. 159-166.

[AD] Atzeni, P., DeBernadis, M.C., "A New Basis for the Weak Instance Model," *Proc. ACM PODS 1987*, pp. 79-86.

[ADM] Ausiello, G., D'Atri, A., Moscarini, M., "Minimal Coverings of Acyclic Database Schemata," *Advances in Database Theory, Vol. 2*, (H. Gallaire, J. Minker and J.M. Nicolas, eds), Plenum Press, 1984, pp. 27-52.

[BFMY] Beeri, C., Fagin, R., Maier, D., Yannakakis, M., "On the Desirability of Acyclic Database Schemes," *Journal of ACM 30:3 (1983)*, pp. 479-513.

[BMSU] Beeri, C., Mendelzon, A.O., Sagiv, Y., Ullman, J.D., "Equivalence of Relational Database Schemes," *SIAM J. on Computing 10:2 (1981)*, pp. 352-370.

[B] Berge, C., *Graphs and Hypergraphs,* North-Holland, Amsterdam, The Netherlands, 1973.

[Bern] Bernstein, P.A., "Synthesizing Third Normal Form Relations from Functional Dependencies," *ACM TODS 1:4 (1976)*, pp. 277-298.

[BBSK] Biskup, J., Bruggemann, H.H., Schnetgoke, L., Kramer, M., "One Flavor Assumption and γ-acyclicity for Universal Relation Views," *Proc. ACM PODS 1986*, pp. 148-159.

[BDB] Biskup, J., Dayal, U., Bernstein, P.A., "Synthesizing Independent Database Schemes," *Proc. ACM SIGMOD 1979*, pp. 143-151.

[C1] Chan, E.P.F., "Optimal Computation of Total Projections with Unions of Simple Chase Join Expressions," *Proc. ACM SIGMOD 1984*, pp. 149-163.

[C2] Chan, E.P.F., "On Finding Unique Minimal Connections for γ-acyclic Schemes," *TR-86-01*, 1986, Department of Computing Science, The University of Alberta.

[CA] Chan, E.P.F., Atzeni, P., "Connection-trap-free Database Schemes," submitted to publication, 1986.

[CH1] Chan, E.P.F., Hernández, H.J., "On the Desirability of γ-acyclic BCNF database schemes," to appear in *Theoretical Computer Science*, 1988.

[CH2] Chan, E.P.F., Hernández, H.J., "Generating Database Schemes Bounded or Constant-time-maintainable by Extensibility," to appear in *Acta Informatica*, 1988.

[CH3] Chan, E.P.F., Hernández, H.J., "Testing Unboundedness of Database Schemes and Functional Dependencies," to appear in *Information Processing Letters*, 1988.

[CM1] Chan, E.P.F., Mendelzon, A.O., "Independent and Separable Database Schemes," *SIAM J. on Computing 16:5 (1987)*, pp. 841-851.

[CM2] Chan, E.P.F., Mendelzon, A.O., "Answering Queries on the Embedded-complete Database Schemes," *Journal of ACM 34:2 (1987)*, pp. 349-375.

[Cod1] Codd, E.F., "A Relational Model for Large Shared Data Banks," *CACM 13:6 (1970)*, pp. 377-387.

[Cod2] Codd, E.F., "Further Normalization of the Data Base Relational Model," *Data Base Systems*, (R. Rustin ed.), pp. 33-64, Prentice-Hall, 1972.

[Cod3] Codd, E.F., "Recent Investigation in Relational Systems," *IFIP Conference, 1974*, pp. 1017-1021.

[CKS] Cosmadakis, S.S., Kanellakis, P.C., Spyratos, N., "Partition Semantics for Relations," *Journal of Computer and System Sciences 33:1 (1986)*, pp. 203-233.

[DM] D'Atri, A., Moscarini, M., "Acyclic Hypergraphs: Their Recognition and Top-down vs. bottom-up generation," *IASI-CNR*, Rome, Italy, 1982.

[F1] Fagin, R., "Multivalued Dependencies and a New Normal Form for Relational Databases," *ACM TODS 2:3 (1977)*, pp. 262-278.

[F2] Fagin, R., "Normal Forms and Relational Database Operators," *Proc. ACM SIGMOD 1979*, pp. 153-160.

[F3] Fagin, R., "Hypergraphs and Relational Database Schemes," *Journal of ACM 30:3 (1983)*, pp. 514-550.

[GM] Graham, M.H., Mendelzon, A.O., "The Power of Canonical Queries," unpublished manuscript, 1983.

[GMV] Graham, M.H., Mendelzon, A.O., Vardi, M.Y., "Notions of Dependency Satisfaction," *Journal of ACM 33:1 (1986)*, pp. 105-129.

[GW] Graham, M.H., Wang, K., "Constant Time Maintenance or the Triumph of the fd," *Proc. ACM PODS 1986*, pp. 202-216.

[GY] Graham, M.H., Yannakakis, M., "Independent Database Schemas," *Journal of Computer and System Sciences 28:1 (1984)*, pp. 121-141.

[H1] Honeyman, P., "Extension Joins," *Proc. VLDB 1980*, pp. 239-244.

[H2] Honeyman, P., "Testing Satisfaction of Functional Dependencies," *Journal of ACM 29:3 (1982)*, pp. 668-677.

[IIK] Ito, M., Iwasaki, M., Kasami, T., "Some Results on the Representative Instance in Relational Databases," *SIAM J. on Computing 14:2 (1985)*, pp. 334-354.

[Ma] Maier, D., *The Theory of Relational Databases*, Computer Science Press, Rockville, 1983.

[MMS] Maier, D., Mendelzon, A.O., Sagiv, Y., "Testing Implications of Data Dependencies," *ACM TODS 4:4 (1979)*, pp. 455-469.

[MRW] Maier, D., Rozenshtein, D., Warren, D.S., "Windows Functions," *Advances in Computing Research, Vol. 3*, JAI Press, 1986, pp. 213-246.

[MUV] Maier, D., Ullman, J.D., Vardi, M.Y., "On the Foundations of the Universal Relation Model," *ACM TODS 9:2 (1984)*, pp. 283-308.

[M] Mendelzon, A.O., "Database States and Their Tableaux," *ACM TODS 9:2 (1984)*, pp. 264-282.

[S1] Sagiv, Y., "Can We Use The Universal Instance Assumption Without Using Nulls?" *Proc. ACM SIGMOD 1981*, pp. 108-120.

[S2] Sagiv, Y., "A Characterization of Globally Consistent Databases and Their Correct Access Paths," *ACM TODS 8:2 (1983)*, pp. 266-286.

[S3] Sagiv, Y., "Evaluation of Queries in Independent Database Schemes," submitted to publication, 1984.

[S4] Sagiv, Y., "On Bounded Database Schemes and Bounded Horn-Clause Programs," *SIAM J. on Computing 17:1 (1988)*, pp. 1-22.

[U] Ullman, J.D., *Principles of Database Systems*, 2^{nd} edition, Computer Science Press, Rockville, 1982.

[Y1] Yannakakis, M., "Algorithms for Acyclic Database Schemes," *Proc. VLDB 1981*, pp. 82-94.

[Y2] Yannakakis, M., private communication, 1981 cited in [F3].

[Z] Zaniolo, C., "A New Normal Form for the Design of Relational Database Schemata," *ACM TODS 7:3 (1982)*, pp. 489-499.

Printing Requisition / Graphic Services

14144

1. **Please complete unshaded areas** on form as applicable.
2. **Distribute copies** as follows: White and Yellow to Graphic Services. **Retain Pink Copies for your records.**
3. On completion of order the **Yellow copy will be returned** with the printed material.
4. **Please direct enquiries, quoting requisition number and account number, to extension 3451.**

TITLE OR DESCRIPTION: CS-88-18 **Independence-reducible Database Schemes**

DATE REQUISITIONED: May 3/88

DATE REQUIRED: ASAP

ACCOUNT NO.: 1 2 6 6 1 1 3 4 1

REQUISITIONER – PRINT: E. Chan

PHONE: 4439

SIGNING AUTHORITY: S. DeAngelis / E. Chan

MAILING INFO – NAME: Sue DeAngelis | DEPT.: C.S. | BLDG. & ROOM NO.: DC 2314 | [X] DELIVER [ ] PICK-UP

Copyright: I hereby agree to assume all responsibility and liability for any infringement of copyrights and/or patent rights which may arise from the processing of, and reproduction of, any of the materials herein requested. I further agree to indemnify and hold blameless the University of Waterloo from any liability which may arise from said processing or reproducing. I also acknowledge that materials processed as a result of this requisition are for educational use only.

NUMBER OF PAGES: 36

NUMBER OF COPIES: 80

TYPE OF PAPER STOCK: [X] BOND [ ] NCR ____ PT. [X] COVER [ ] BRISTOL [X] SUPPLIED [ ] ____

PAPER SIZE: [X] 8½ x 11 [ ] 8½ x 14 [ ] 11 x 17 [ ] ____

PAPER COLOUR: [X] WHITE [ ] ____ INK: [X] BLACK [ ] ____

PRINTING: [ ] 1 SIDE ____ PGS. [X] 2 SIDES ____ PGS. NUMBERING FROM ____ TO ____

BINDING/FINISHING: [X] COLLATING [X] STAPLING [ ] ____ PUNCHED [ ] PLASTIC RING — 3 down left side

FOLDING/PADDING: CUTTING SIZE:

Special Instructions

Math fronts and backs enclosed

COPY CENTRE — OPER. NO. | BLDG. | MACH. NO.

DESIGN & PASTE-UP — OPER. NO. | TIME | LABOUR CODE

D 0 1
D 0 1
D 0 1

TYPESETTING — QUANTITY

PAP 0 0 0 0 0 0 — T 0 1
PAP 0 0 0 0 0 0 — T 0 1
PAP 0 0 0 0 0 0 — T 0 1

PROOF

PRF
PRF
PRF

NEGATIVES — QUANTITY | OPER. NO. | TIME | LABOUR CODE

FLM — C 0 1
FLM — C 0 1
FLM — C 0 1
FLM — C 0 1
FLM — C 0 1

PMT

PMT — C 0 1
PMT — C 0 1
PMT — C 0 1

PLATES

PLT — P 0 1
PLT — P 0 1
PLT — P 0 1

STOCK

0 0 1
0 0 1
0 0 1
0 0 1

BINDERY

RNG — B 0 1
RNG — B 0 1
RNG — B 0 1
MIS 0 0 0 0 0 0 — B 0 1

OUTSIDE SERVICES

$ ____ COST

TAXES – PROVINCIAL [ ] FEDERAL [ ]

GRAPHIC SERV. OCT. 85 482-2

To Sue

From Tae

Date April 29 88

memo

University of Waterloo

Sue!

Please order 80 copies of the attach. report - I've attached the front and back covers of the T.R style we would like.

Also, would you leave me a copy of the req'n in my mbox? Thanks.

His A/c: 126-6113-41.

T.