A Characterization of Constant-time-maintainability for BCNF Database Schemes

Héctor J. Hernández
Department of Computer Science
Texas A&M University
College Station, Texas, USA 77843-3112

Edward P.F. Chan
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

March 9, 1988

Abstract

The *maintenance problem* (for database states) of a database scheme **R** with respect to a set of functional dependencies F is the following decision problem: Let **r** be a consistent state of **R** with respect to F and assume we insert a tuple t into $r_p \in \mathbf{r}$. Is $\mathbf{r} \cup \{t\}$ a consistent state of **R** with respect to F? **R** is said to be *constant-time-maintainable* with respect to F if there is an algorithm that solves the maintenance problem of **R** with respect to F in time independent of the state size.

A characterization of constant-time-maintainability for the class of BCNF database schemes is given. An efficient algorithm that tests this characterization is shown, as well as an algorithm for solving the maintenance problem in time independent of the state size. It is also proven that constant-time-maintainable BCNF database schemes are bounded. In particular, it is shown that total projections of the representative instance can be computed via unions of projections of extension joins. Throughout we assume that database schemes are cover embedding and BCNF, and that functional dependencies are given in the form of key dependencies.

1 Introduction

An important task in the design theory is to identify classes of good database schemes. Codd [Co] observed that in the presence of functional dependencies, certain anomalies may exist when a relation is being updated. He proposed normalization to avoid the anomaly problem. Since then, several normal forms have been defined and studied. Among these normal forms, Boyce-Codd Normal Form (BCNF) is considered to be one of the most important normal forms. Although there are problems with the construction of BCNF database schemes in general [BB, BG, LO, O], LeDoux and Parker [LeP] showed that under certain reasonable assumptions, BCNF database schemes are free of the anomaly problem. Moreover, they argued that the construction problem with BCNF database schemes does not really exist in most real-life applications. With functional dependencies as the constraints imposed on a database scheme, BCNF is a good design goal toward which a database designer should strive, since this class of schemes seems to capture naturally the principle of separation stated in [BBG], and therefore allowing independent facts to be stored in separate relations.

Recent work on design theory concentrates on some other desirable properties. Query answering and constraint enforcement are supposed to be two major and very important functions in a database system. Consequently, designing a database scheme that facilitates easy and efficient query answering and constraint enforcement is highly desirable. Some work has been done on this problem and several classes of database schemes with such properties were identified [AC, CA, CH1, CM, IIK, S3].

Recently, Graham and Wang [GW] proposed the notion of constant-time-maintainability to capture

the intuition on efficient constraint enforcement. Informally, a database scheme is constant-time-maintainable if there is an algorithm which determines whether any of its consistent states plus an inserted tuple is consistent by examining at most k tuples in the state, where k is a constant independent of the state size. The idea behind constant-time-maintainability is that with the inserted tuple, the algorithm knows *exactly* which tuples in the consistent state need to be examined to determine consistency of the updated state. To illustrate this idea, let us consider the following example.

Example 1: Let $\mathbf{R} = \{$*Affiliated_to*(*Instructor*, *Dept*), *Coordinate*(*Instructor*, *Course*), *Offered_by*(*Course*, *Dept*)$\}$ be a database scheme. We assume each instructor is associated with a unique department and coordinates at most one course. Moreover, each course is offered by a unique department and is coordinated by at most one instructor in the department. With these constraints, all attributes, except *Dept*, are keys of the corresponding relations. The set of functional dependencies $F=\{$*Instructor* $\rightarrow$ *Dept*, *Instructor* $\rightarrow$ *Course*, *Course* $\rightarrow$ *Instructor*, *Course* $\rightarrow$ *Dept*$\}$ models these constraints. Clearly, $\mathbf{R}$ is BCNF with respect to F. From the result in this paper, we will show that $\mathbf{R}$ is constant-time-maintainable with respect to F.

To see how the maintenance problem can be solved for this application, let us consider a tuple $t =< i, d >$ on *Affiliated_to* being inserted into a consistent state $\mathbf{r}$ on $\mathbf{R}$. First, we verify that the inserted tuple with the existing *Affiliated_to* relation satisfy the key dependency *Instructor* $\rightarrow$ *Dept*. If not, the updated state is not consistent and we stop. By the assumption that *Affiliated_to* is a BCNF scheme and $\mathbf{r}$ is a consistent state on $\mathbf{R}$, only one tuple needs to be examined during this step. Having proven satisfaction within the *Affiliated_to* relation, we need to ensure that the department of the course the instructor 'i' coordinates, if it exists, is 'd'. This is accomplished by retrieving a tuple from the *Coordinate* relation with the *Instructor*-component equal to 'i'. Let us denote this tuple by u, if it exists. We then retrieve another tuple, call it w, from the *Offered_by* relation with its *Course*-component equal to the *Course*-component of u. If any of these tuples does not exist, the state is consistent. Otherwise let $u =< i, c >$ and $w =< c, d_1 >$ be the retrieved tuples. If 'd_1' is not equal to 'd', then the updated state is not consistent. Otherwise it is. □

In the above algorithm, given an inserted tuple on *Affiliated_to*, we know exactly which tuples we should look for in determining the consistency of the updated state. As we will show later, this algorithm correctly determines if an inserted tuple on the *Affiliated_to* relation results in a consistent state or not. For insertions on other relations, similar algorithms can be derived.

The class of constant-time-maintainable database schemes was proposed by Graham and Wang [GW] as a generalization of independent schemes [GY, IIK, S3]. An exponential-time algorithm was given by them to recognize the class of constant-time-maintainable database schemes when a full join dependency and a set of functional dependencies are given. The recognition of constant-time-maintainable database schemes for the case when only functional dependencies are considered is a very difficult open problem. Some work has been done on this problem. The class of independent database schemes is constant-time-maintainable by definition [GY, IIK, S3]. For BCNF database schemes, there are three distinct proper subclasses which are known to be constant-time-maintainable: Sagiv's BCNF independent schemes [S2] are constant-time-maintainable by definition, γ-acyclic BCNF database schemes were proven to be constant-time-maintainable in [CH1], and key-equivalent split-free BCNF database schemes were shown to be constant-time-maintainable in [CH2]. Also, in [BV], Brossda and Vossen characterized a class of constant-time-maintainable database schemes. However, since they use the foreign-key constraint [S1], they have to check that a consistent state satisfies this constraint after a deletion, and to do this, their algorithm takes time proportional to the size of the state.

In this paper, we give a characterization of constant-time-maintainability for the entire class of BCNF database schemes. We show that constant-time-maintainability for BCNF database schemes can be characterized efficiently. An algorithm is also found for solving the maintenance problem. We also prove that this class of schemes is bounded and, more interestingly, that total projections can be computed via unions of projections of extension joins. Consequently, not only BCNF constant-time-maintainable database schemes are desirable with respect to constraint enforcement, but also they are desirable with respect to query answering. This also shows that the class of schemes we characterize is the largest known class of constant-time-maintainable database schemes which is bounded.

In Section 2, we give most of the definitions needed for this paper. In Section 3, we charac-

terize when a BCNF database scheme is constant-time-maintainable and show that the characterization can be tested efficiently. Moreover, an algorithm is given to find the constant number of tuples required in solving the maintenance problem. In Section 4, we show that unions of projections of extension joins can compute total projections in the representative instance. Finally, we give our conclusions in Section 5.

2 Definitions and Notation

2.1 Basic Definitions

We shall follow standard notation [Ma, U] and only give some non-standard definitions here. The kind of constraints considered here is functional dependencies (fd's). Unless otherwise stated, we assume that database schemes are cover embedding BCNF, and that fd's are given in the form of key dependencies.

Given a tableau T [ASU], we shall denote the final tableau in a chase of T with respect to (wrt) a set of fd's F as $CHASE_F(T)$ [ABU, MMS]. In this paper, we are working under the *weak instance model* [GMV, H2, M, S1, V, Y]. Given two database states of $\mathbf{R}$, $\mathbf{r}=< r_1, \ldots, r_k >$ and $\mathbf{s}=< s_1, \ldots, s_k >$, we say that $\mathbf{r} \subseteq \mathbf{s}$ if $r_i \subseteq s_i$, for all $1 \leq i \leq k$. We say that $\mathbf{r}$ is a *proper substate* of s if $\mathbf{r} \subseteq$ s and for some $1 \leq i \leq k$, $r_i \subset s_i$. Also we define $\mathbf{s} \cup \mathbf{r} = < s_1 \cup r_1, \ldots, s_k \cup r_k >$. Now let s'_j be a relation on $S_j \in \mathbf{R}$ and let s_j be the relation on S_j in the state s. Then $\mathbf{s} - s'_j$ shall denote s with s_j replaced by $s_j - s'_j$, and $\mathbf{s} \cup s'_j$ shall denote s with s_j replaced by $s_j \cup s'_j$. Let $\mathbf{S}=\{S_1, \ldots, S_m\}$ be a subset of $\mathbf{R}$. We say that $\bowtie_{j=1}^{m} S_j$ is an *extension join (of S_1)* if for all $1 \leq j < m$, $S_1 \cup \ldots \cup S_j$ contains a key of S_{j+1} [H1]. We say that $\bowtie_{j=1}^{m} S_j$ *covers* X if $S_1 \cup \ldots \cup S_m \supseteq X$.

2.2 Independence and Boundedness

Let $\mathbf{R}=\{R_1, \ldots, R_k\}$ be a (not necessarily BCNF) database scheme and let $F = F_1 \cup \ldots \cup F_k$, where F_i is the set of key dependencies embedded in R_i, for all $R_i \in \mathbf{R}$. Then $\mathbf{R}$ is *independent wrt F* 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]. Independent schemes were investigated also in [GY, IIK, S3].

A database scheme $\mathbf{R}$ is *bounded* wrt F if every total tuple t in the representative instance of any consistent state $\mathbf{r}$ of $\mathbf{R}$ wrt F can be computed by a predetermined relational expression [GM, MUV].

2.3 Constant-time-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 $\mathbf{r} \cup \{t\}$ a consistent state of $\mathbf{R}$ wrt F? We say that $< \mathbf{r}, t >$ 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.

Following [GW], we define constant-time-maintainable schemes. Suppose there is an algorithm $\mathbf{A}$ that solves the maintenance problem of $\mathbf{R}$ wrt F. Assume further that $\mathbf{r}$ is stored on a device that responds to requests of the form $< R_i, \Psi >$, where $R_i \in \mathbf{R}$ and Ψ is a boolean combination of formula of the form $A =$ 'a', where $A \in R_i$, a is an element of the domain of A, and a must be a value that comes from the inserted tuple or from tuples retrieved in requests previously issued by $\mathbf{A}$ (this very last condition is called the *no guessing assumption*). The storage device responds to a request by returning, if it exists, an arbitrary tuple from $r_i \in \mathbf{r}$ that satisfies Ψ. For any instance $< \mathbf{r}, t >$ of the maintenance problem of $\mathbf{R}$ wrt F, we define $\#\mathbf{A}(\mathbf{r}, t)$ to be the number of requests, of the above form, made by $\mathbf{A}$ on the instance $< \mathbf{r}, t >$. We say that $\mathbf{A}$ solves the maintenance problem of $\mathbf{R}$ wrt F in *constant time* if there is a constant k, k is independent of the state size and $k \geq 0$, such that $k \geq \#\mathbf{A}(\mathbf{r}, t)$, for all instances $< \mathbf{r}, t >$ of the maintenance problem of $\mathbf{R}$ wrt F. A database scheme $\mathbf{R}$ is said to be *constant-time-maintainable (ctm)* wrt F if there is an algorithm that solves the maintenance problem of $\mathbf{R}$ wrt F in constant time [GW].

2.4 Key-equivalent Schemes

Let $\mathbf{R}$ be a database scheme and let F be the set of key dependencies embedded in $\mathbf{R}$. Then $\mathbf{R}$ is said to be *key-equivalent* wrt F if for all R_i in $\mathbf{R}$, $R_i^+ = \cup \mathbf{R}$. Let $\mathbf{R}$ be key-equivalent wrt F, let K be a key in $\mathbf{R}$, and let $R_l \in \mathbf{R}$. Then K is *split* in R_l^+ *(wrt F)* if there is a partial computation of R_l^+ that covers K but none of the schemes in such a computation contains K. Then we say that R_l is *split-free (wrt F)* if no key is split in R_l^+; else it is *split*. If $\mathbf{R}$ is key-equivalent (wrt F), then we say that $\mathbf{R}$ is *split-free (wrt F)* if for all $R_i \in \mathbf{R}$, R_i is split-free (wrt F); else it is *split*.

The following are results from [CH2] about key-equivalent and split-free database schemes that we

need later on in the paper.

Theorem 1: Let **S** be a split-free key-equivalent database scheme wrt F, where F is the set of key dependencies embedded in **S**. Let **s** be a consistent state on **S** wrt F. Let s, a tuple t on some $S_i \in \mathbf{S}$, and F be the input to Algorithm 1 shown below. Algorithm 1 outputs *yes* exactly when $\mathbf{s} \cup \{t\}$ is consistent wrt F.

Algorithm 1

Input: s, t, and F, where s is a consistent state for **S** wrt F, F is the set of key dependencies embedded in **S**, **S** is split-free and key-equivalent wrt F, and t is a tuple on S_i, where $S_i \in \mathbf{S}$.

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

Method:

(1) Let $K_1, \ldots, K_u$ be the set of keys of S_i. For each K_j, invoke Algorithm 2 (shown in Section 3.1) with input **s** and $t[K_j]$. Let t'_j on C_j be the tuple returned by Algorithm 2 when **s** and $t[K_j]$ are used as input, for all $1 \leq j \leq u$;

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

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

Theorem 2: Let **S** be a key-equivalent database scheme wrt F, where F is the set of key dependencies embedded in **S**. Then **S** is ctm wrt F if and only if **S** is split-free wrt F.

2.5 Independence-reducibility

A *partition* of a set S is a collection of 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*.

Let $\mathbf{R}=\{R_1, \ldots, R_n\}$ be a database scheme and let F be the set of key dependencies embedded in **R**. In this subsection, **R** need not be in BCNF. **R** is said to be *(key-equivalent) independence-reducible wrt* F, if there is a partition $\mathbf{T}=\{T_1, \ldots, T_k\}$ of **R** such that (a) $\mathbf{D} = \{\cup\, T_p | T_p \in \mathbf{T}\}$ 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.

Let **R** be a database scheme, let F be the set of key dependencies embedded in **R**, and let $R_i \in \mathbf{R}$. Define $\mathbf{R}_i^*$ as the largest subset of **R** containing R_i such that $\mathbf{R}_i^*$ is key-equivalent wrt its embedded key dependencies. $\{\mathbf{R}_i^* | R_i \in \mathbf{R}\}$ is a partition of **R** and is unique; $\{\mathbf{R}_i^* | R_i \in \mathbf{R}\}$ is called the *key-equivalent partition* of **R** *(wrt F)*. The output from *KEP*, shown below, when **R** and F are its input is the key-equivalent partition of **R** [CH2]. Let $\mathbf{T}=\{T_1, \ldots, T_k\}$ be the key-equivalent partition of **R** wrt F. Then we say that **R** is *split-free (wrt F)* if for each T_i in **T**, T_i is split-free wrt G_i, where G_i is the set of key dependencies which are embedded in schemes in T_i.

---

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

Input: A database scheme $\mathbf{R}=\{R_1, \ldots, R_n\}$ and F, where F is the set of key dependencies embedded in **R**.

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

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

Method:

(1) **begin**

(2) Let *part* $=\{\, [R_j] | R_j \in \mathbf{R}\}$;

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

(4) **end**

The following are results from [CH2] about independence-reducible database schemes that we need later on in the paper.

Theorem 3: Let $\mathbf{R}=\{R_1, \ldots, R_n\}$ be a database scheme and let F be the set of key dependencies embedded in **R**. Then there is a polynomial-time algorithm that determines whether **R** is independence-reducible wrt F.

Theorem 4: Let **R** be an independence-reducible database scheme wrt F, where F is the set of key dependencies embedded in **R**. Then (a) **R** is ctm wrt F if and only if **R** is split-free wrt F; (b) **R** is bounded wrt F.

3 Ctm BCNF Schemes

In this section, we give an efficient characterization of constant-time-maintainability for BCNF database schemes and derive an algorithm for solving the maintenance problem. In Section 3.1, we first prove that a BCNF database scheme $\mathbf{R}$ is ctm implies $\mathbf{R}$ is independence-reducible. This shall imply $\mathbf{R}$ is ctm if and only if $\mathbf{R}$ is independence-reducible and split-free by Theorem 4(a) above. In Section 3.2, we present an efficient algorithm that recognizes exactly the class of ctm BCNF database schemes. In Section 3.3, we derive an algorithm and show how the maintenance problem can be solved incrementally. (Due to space restrictions we omit most of the proofs in this section.)

3.1 Proof of the Main Theorem

In this subsection we want to prove that if $\mathbf{R}$ is BCNF and ctm wrt F, where F is the set of key dependencies embedded in $\mathbf{R}$, then $\mathbf{R}$ is independence-reducible wrt F. We prove this fact by showing that given a BCNF database scheme $\mathbf{R}$, $\mathbf{R}$ is not independence-reducible wrt F implies $\mathbf{R}$ is not ctm wrt F. The proof involves a construction of a rather complex database state on a non-independence-reducible database scheme $\mathbf{R}$, and this state demonstrates that $\mathbf{R}$ cannot be ctm.

Let $\mathbf{R}$ be a BCNF database scheme wrt F, where F is the set of key dependencies embedded in $\mathbf{R}$. Assume $\mathbf{R}$ is not independence-reducible wrt F. Let $\mathbf{T} = \{KE_1, \ldots, KE_n\}$ be the key-equivalent partition of $\mathbf{R}$. Let $\mathbf{D} = \{D_1 = \cup KE_1, \ldots, D_n = \cup KE_n\}$. Since $\mathbf{R}$ is not independence-reducible wrt F and by definition of independence-reducibility, $\mathbf{D}$ is not independent wrt F. We can prove that $\mathbf{D}$ is not independent wrt F implies that there are R_p and R_q, $R_p \in KE_i$ and $R_q \in KE_j$, $i \neq j$, $1 \leq i, j \leq n$, such that $(R_p)^+_{F-F_q} \supseteq K_qA$, where $K_q \rightarrow A \in F_q^+$, $A \notin K_q$, and F_q is the set of key dependencies embedded in R_q. Observe that this is a violation of independence that involves schemes from at least two different blocks of $\mathbf{T}$.

Let us assume without loss of generality that $R_1, \ldots, R_{l-1}$ are the schemes in a computation of $(R_p)^+_{F-F_q}$ such that $R_1 = R_p$, $K_qA \subseteq R_1 \cup \ldots \cup R_{l-1}$, but $K_qA \not\subseteq R_1 \cup \ldots \cup R_{l-2}$, and let $R_l = R_q$; we can prove that $l \geq 3$, that is, the above sequence exists and is well-defined. We now use $R_1, \ldots, R_l$ to define an inconsistent state $\mathbf{r}$ on $\mathbf{R}$. Let z be a tuple on $U_l = R_1 \cup \ldots \cup R_l$ such that $z[B]$ is a distinct constant for each $B \in U_l$. Then $\mathbf{r}$ is the state defined as follows: For all $R_i \in \mathbf{R} - \{R_1, \ldots, R_l\}$, $r_i = \emptyset$; for all $R_i \in \{R_1, \ldots, R_{l-1}\}$, $r_i = \{t_i\}$, where $t_i = z[R_i]$; $r_l = \{t_l\}$, where $t_l[R_l - \{A\}] = z[R_l - \{A\}]$, and $t_l[A]$ is a distinct constant that is different from $z[A]$.

Lemma 1: $\mathbf{r}$ is inconsistent wrt F.

In the rest of this subsection, t_i shall denote the above defined tuple on R_i, for $1 \leq i \leq l$.

Let $\mathbf{r}_{min}$ be a minimal inconsistent substate of $\mathbf{r}$, that is, let $\mathbf{r}_{min} \subseteq \mathbf{r}$ be such that $\mathbf{r}_{min}$ is inconsistent wrt F and any of its proper substates is consistent wrt F. The state $\mathbf{r}_{min}$ can be obtained from $\mathbf{r}$ by removing one by one the nonempty singleton relations in $\mathbf{r}$ while the resultant proper substate is still inconsistent.

Let $\mathbf{R}_{ne} = \{R_k \in \mathbf{R} | r_k \text{ in } \mathbf{r}_{min} \text{ is nonempty}\}$; that is, $\mathbf{R}_{ne}$, where ne stands for 'nonempty', is the set of schemes on which $\mathbf{r}_{min}$ has nonempty relations. Now let F_{ne} be the set of key dependencies embedded in $\mathbf{R}_{ne}$.

Lemma 2: There exists $R_v \in \mathbf{R}_{ne}$ such that $\{R_k | R_k \in \mathbf{R}_{ne} \text{ but } R_k \not\subseteq (R_v)^+_{F_{ne}}\} \neq \emptyset$.

Consider the extension algorithm, Algorithm 2, shown below. We are going to let $\mathbf{r}_{min}$ and t_v, where $r_v = \{t_v\}$ is the relation on the scheme given by Lemma 2, be the input to Algorithm 2; since $\mathbf{r}_{min}$ is inconsistent wrt F, Algorithm 2 with this input may not be well-defined; we can prove that this is not the case.

The following is a consequence of the above lemma.

Corollary 1: There is some $r_v = \{t_v\} \in \mathbf{r}_{min}$ such that $\{R_k | R_k \in \mathbf{R}_{ne} \text{ but } R_k \not\subseteq C_v\} \neq \emptyset$, where C_v is the set of attributes output from Algorithm 2 when $\mathbf{r}_{min}$ and t_v are its input.

Let t_v be the tuple on R_v given by Corollary 1, and let **Rest'** be **Rest** after the execution of the **while**-loop in Algorithm 2 when $\mathbf{r}_{min}$ and t_v are its input. Let $\mathbf{R}_v = \mathbf{R} - \mathbf{Rest'}$; now notice that $\mathbf{R} - \mathbf{Rest'} = \mathbf{R}_{ne} - \mathbf{Rest'}$ since if $S_j \in \mathbf{R} - \mathbf{Rest'}$, then $S_j \in \mathbf{R}_{ne}$. Thus $\mathbf{R}_v = \mathbf{R}_{ne} - \mathbf{Rest'}$. Now let $\mathbf{R}_{ne\_v} = \mathbf{R}_{ne} - \mathbf{R}_v$; by Corollary 1, $\mathbf{R}_{ne\_v} \neq \emptyset$. $\mathbf{R}_v$ is nonempty, since at least $R_v \in \mathbf{R}_v$.

Now we are going to define two nonempty proper substates of $\mathbf{r}_{min}$, one with nonempty relations on $\mathbf{R}_{ne\_v}$ and one with nonempty relations on $\mathbf{R}_v$. Let $\mathbf{r}_{ne\_v}$ be the following proper substate of $\mathbf{r}_{min}$: For all $R_i \in \mathbf{R} - \mathbf{R}_{ne\_v}$, the relation on R_i is empty; for all $R_i \in \mathbf{R}_{ne\_v}$, the relation on R_i is set equal to $r_i = \{t_i\} \in \mathbf{r}_{min}$. Let $\mathbf{r}_v$ be the following proper substate of $\mathbf{r}_{min}$: For all $R_i \in \mathbf{R} - \mathbf{R}_v$, the relation on R_i is empty; for all $R_i \in \mathbf{R}_v$, the relation on R_i is set equal to $r_i = \{t_i\} \in \mathbf{r}_{min}$.

Lemma 3: Let $\mathbf{r}_v$ and $\mathbf{r}_{ne\_v}$ be as defined above.

Algorithm 2

Input: A state **s** on some database scheme $\mathbf{S} = \{S_1, \ldots, S_m\}$, and a total tuple t on K, where $K \subseteq U$, $U = \cup\mathbf{S}$.

Output: t' and C, where t' is a total tuple on C, $C \subseteq U$.

Comment: t' is t extended with constants from **s**.

Method:

(1) Let $t' = t$, $C = K$, and $t'[B]$ is a distinct nondistinguished variable for all $B \in U - K$;

(2) Let **Rest** = **S**;

(3) **while** (there are $S_j \in$ **Rest**, a tuple p in $s_j \in \mathbf{s}$, and a key K_j of S_j such that $p[K_j] = t'[K_j]$)

do $t'[S_j] = p[S_j]$; $C = C \cup S_j$;
Rest = **Rest** $- \{S_j\}$; **end**

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

Then $\mathbf{r}_v$ and $\mathbf{r}_{ne\_v}$ are nonempty proper substates of $\mathbf{r}_{min}$.

The following are facts that we need about t_v^*.

Lemma 4: Let t_v be as given by Corollary 1, and let t_v^* and C_v be the output from Algorithm 2 when $\mathbf{r}_{min}$ and t_v are its input. Then (a) $t_v^*[B] = t_h[B]$, for some $r_h = \{t_h\} \in \mathbf{r}_{ne\_v}$ and some $B \in C_v \cap R_h$; (b) let $r_1 = \{t_1\}, \ldots, r_y = \{t_y\}$ be the nonempty relations in $\mathbf{r}_{ne\_v}$ such that for all $1 \leq h \leq y$, $t_v^*[B_h] = t_h[B_h]$, for some $B_h \in R_h \cap C_v$; let, for all $1 \leq h \leq y$, V_h be $\{B | B \in R_h \cap C_v$, and $t_v^*[B] = t_h[B]\}$; then we have that for all $1 \leq h \leq y$, V_h is not a super key of R_h, and $V_h \subset R_h$.

Now, for $1 \leq h \leq y$, let R_h and V_h be as defined in Lemma 4(b) above, and let $\mathbf{r}_h^*$ be a state on **R** whose only nonempty relation is $r_h^* = \{u | u$ is a tuple on R_h such that $u[V_h] = t_h[V_h]$, and for all $B \in R_h - V_h$, $u[B]$ is a distinct constant$\}$, where r_h^* has an arbitrary but finite number of tuples. Lemma 4(b) guarantees that r_h^* can have any number of tuples with the same value on V_h and, more importantly, that r_h^* is consistent wrt the key dependencies embedded in R_h. Assume $R_{y+1}, \ldots, R_z$ are the schemes (if any) in $\mathbf{R}_{ne\_v} - \{R_1, \ldots, R_y\}$. Then for $y + 1 \leq h \leq z$, let $\mathbf{r}_h^*$ be a state on **R** whose only nonempty relation is $r_h^* = \{u | u$ is a tuple on R_h such that for all $B \in R_h$, $u[B]$ is a distinct constant$\}$, where r_h^* has an arbitrary but finite number of tuples.

Lemma 5: Let $\mathbf{r}_1^*, \ldots, \mathbf{r}_z^*$ be as defined above. Then $\mathbf{r}_0^* \cup \mathbf{r}_1^* \cup \ldots \cup \mathbf{r}_z^* - r_j$, where $\mathbf{r}_0^* = \mathbf{r}_{min}$ and $r_j = \{t_j\}$ is any nonempty relation in $\mathbf{r}_{min}$, is consistent wrt F.

Suppose now that **R** is ctm wrt F. Then there is an algorithm **A** that can solve the maintenance problem of **R** wrt F by issuing no more than c requests, for some constant $c \geq 1$, and c is independent of the state size. Let t_v be the tuple on R_v given by Corollary 1, and let t_v^* and C_v be the output from Algorithm 2 when $\mathbf{r}_{min}$ and t_v are its input.

Let $\mathbf{r}^\# = \mathbf{r}_0^* \cup \mathbf{r}_1^* \cup \ldots \cup \mathbf{r}_z^* - \{t_v\}$; where $\mathbf{r}_1^*, \ldots, \mathbf{r}_z^*$ are as defined above but for all $1 \leq h \leq z$, $\mathbf{r}_h^*$ has c tuples. $\mathbf{r}^\#$ is consistent wrt F by Lemma 5. Observe that the relation on R_h, for all $1 \leq h \leq z$, in $\mathbf{r}^\#$ is $r_h \cup r_h^*$, where $r_h = \{t_h\}$ is the relation on R_h in $\mathbf{r}_{min}$, and hence has $c + 1$ tuples. Now let us insert t_v into $\mathbf{r}^\#$. That is let $\mathbf{r}'' = \mathbf{r}^\# \cup \{t_v\}$. Since $\mathbf{r}_{min}$ is a substate of $\mathbf{r}''$ and, by definition, $\mathbf{r}_{min}$ is inconsistent wrt F, $\mathbf{r}''$ is inconsistent wrt F. Lemma 5 implies **A** needs to retrieve all the tuples in $\mathbf{r}_{min}$ and hence, by Lemma 3, it needs to retrieve all the tuples in $\mathbf{r}_{ne\_v}$ to show $\mathbf{r}''$ is inconsistent wrt F. We are going to prove that in the worst case **A** will not retrieve any tuple from $\mathbf{r}_{ne\_v}$ in c requests.

Then with t_v, a tuple in $\mathbf{r}_v$, as the inserted tuple, **A** will eventually try to access the tuples in $\mathbf{r}_{ne\_v}$ via requests of the form $< R_h, \Psi >$, $R_h \in \mathbf{R}_{ne\_v}$. We may assume without loss of generality that before issuing its first request for tuples in $\mathbf{r}_{ne\_v}$, **A** already retrieved all tuples from $\mathbf{r}_v$ and therefore no more requests for tuples from $\mathbf{r}_v$ are needed. Let $< R_{h_1}, \Psi_1 >, \ldots, < R_{h_c}, \Psi_c >$ be any sequence of c requests that **A** may issue to retrieve the tuples in $\mathbf{r}_{ne\_v}$. Let, for $1 \leq i \leq c$, Att_i be the attributes for which values are known to **A** before it issues $< R_{h_i}, \Psi_i >$, and for any $B \in Att_i$, let $Const(B)$ be the set of values known to **A** on B; by our assumption above, $Att_1 = C_v$ and $Const(B) = \{t_v^*[B]\}$, for any $B \in Att_1$.

The following lemma states in its first part that all tuples in any relation in $\mathbf{r}_{ne\_v}$, which are not retrieved before $< R_{h_i}, \Psi_i >$ is issued, stand in the same relationship with the constants known to **A** before $< R_{h_i}, \Psi_i >$ is issued. Its second and third parts are direct consequences of the first one and say that **A** cannot solve the maintenance problem in c requests.

Lemma 6: For $1 \leq i \leq c$, (a) for all $B \in Att_i$, for all $d \in Const(B)$, and for all $R_h \in \mathbf{R}_{ne\_v}$ such that $B \in R_h$, we have that either $d = t[B]$ for all t in r_h

not retrieved before $< R_{h_i}, \Psi_i >$ is issued, or $d \neq t[B]$ for all t in r_h not retrieved before $< R_{h_i}, \Psi_i >$ is issued; (b) if $u_i \in r_{h_i}$ satisfies Ψ_i in $< R_{h_i}, \Psi_i >$, where u_i is a tuple not retrieved before $< R_{h_i}, \Psi_i >$ is issued, then there are at least $c + 2 - i$ tuples in r_{h_i} not retrieved before $< R_{h_i}, \Psi_i >$ is issued that satisfy Ψ_i; and thus (c) (in the worst case for **A**) the tuple in r_{h_i} from $\mathbf{r}_{ne\_v}$ is not retrieved by $< R_{h_i}, \Psi_i >$.

Hence by Lemma 6, in the worst case with c requests **A** will not retrieve any tuple from $\mathbf{r}_{ne\_v}$. Thus **A** does not solve the maintenance problem of R wrt F in constant time. Hence **R** is not ctm wrt F. We have proved the following theorem.

Theorem 5: Let **R** be a BCNF database scheme wrt F, where F is the set of key dependencies embedded in **R**. If **R** is ctm wrt F, then **R** is independence-reducible wrt F.

3.2 Recognition Algorithm

The following theorem tells us how to recognize the class of ctm BCNF database schemes; it follows directly from Theorems 5 and 4(a) above.

Theorem 6: Let **R** be a BCNF database scheme wrt F, where F is the set of key dependencies embedded in **R**. Then **R** is ctm wrt F if and only if R is independence-reducible and split-free wrt F.

The above theorem implies that we have a polynomial-time recognition algorithm for the class of ctm BCNF database schemes.

Corollary 2: Let **R** be a BCNF database scheme wrt F, where F is the set of key dependencies embedded in **R**. There is a polynomial-time algorithm that determines whether or not **R** is ctm wrt F.

Proof Sketch: By Theorem 6, **R** is ctm wrt F if and only if it is independence-reducible and split-free wrt F. By Theorem 3, there is a polynomial-time algorithm that determines whether **R** is independence-reducible wrt F; this algorithm is Algorithm 3, shown below, without step (4). By definition of splitness, step (4) of Algorithm 3 determines if **R** is split-free wrt F. This step can also be done in polynomial time, since we can prove the following. For any KE_i in the key-equivalent partition of **R**, let K be a key in KE_i, let F_i be the set of key dependencies embedded in KE_i, and let $W=\{R_p | R_p \in KE_i \text{ and } K \not\subseteq R_p\}$. Then K is split in R_l^+ wrt F_i, for some $R_l \in KE_i$, if and only if there is a row t_j in $CHASE_{F_i}(T_W)$ such that $t_j[K]$ are all distinguished variables, where T_W is the tableau for W [ABU, ASU]. Then our claim holds. □

Algorithm 3

Input: A database scheme **R** and G the set of key dependencies embedded in **R**, where **R** is BCNF wrt G.

Output: *Accept* or *reject*. If *accept* is output, **R** is ctm wrt G; if *reject* is output **R** is not ctm wrt G.

Method:

(1) Generate the key-equivalent partition $\mathbf{T}=\{KE_1, \ldots, KE_n\}$ of **R** via KEP(**R**, G) (Shown in Section 2);

(2) let F_j be the 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*, and **exit**;

(4) **if** for some $KE_i \in \mathbf{T}$, KE_i is split wrt F_i **then** output *reject*, and **exit**;

(5) output *accept*.

3.3 Maintenance Algorithm

In this section, we describe the maintenance algorithm for ctm BCNF database schemes.

Let **R** be a ctm BCNF database scheme wrt F, where F is the set of key dependencies embedded in **R**. Then by Theorem 6, **R** is independence-reducible and split-free wrt F. Let **r** be a consistent state of **R** wrt F. Suppose $r_m \in \mathbf{r}$ is updated by an insertion of a tuple t. 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.

Let $\mathbf{r}'$ be the following state on T_j: for each $R_i \in T_j$, $r_i \in \mathbf{r}'$ is equal to the relation on R_i in **r**. Let F_j be the set of key dependencies from F which are embedded in schemes in T_j. It should be clear that $\mathbf{r}'$ is consistent wrt F_j. Since we can prove that $\mathbf{r} \cup \{t\}$ is consistent wrt F if and only if $\mathbf{r}' \cup \{t\}$ is consistent wrt F_j, to verify whether $\mathbf{r} \cup \{t\}$ is consistent wrt F all we have to do is to check if $\mathbf{r}' \cup \{t\}$ is consistent wrt F_j. We now show how to do this.

By definition of splitness, **R** is split-free wrt F implies T_j is split-free wrt F_j. Hence by Theorem 2, T_j is ctm wrt F_j. From Theorem 1, Algorithm 1,

shown in Section 2, outputs *yes* with input $\mathbf{r}'$, t, F_j exactly when $\mathbf{r}' \cup \{t\}$ is consistent wrt F_j. It should be clear that Algorithm 1 runs in time independent of the state size.

Example 2: Let us consider the BCNF database scheme in Example 1. By the *KEP* function in Section 2, the key-equivalent partition of $\mathbf{R}$ is $\mathbf{R}$ itself. Hence $\mathbf{R}$ is independence-reducible wrt F. Since all keys in $\mathbf{R}$ are single-attribute keys, $\mathbf{R}$ is split-free. By Theorem 6, $\mathbf{R}$ is ctm wrt F. Let a consistent state r on $\mathbf{R}$ be as follows. $\mathbf{r} = <$*Affiliated_to*$=\{< i, CS >, < j, CS >\}$, *Coordinate*$=\{< i, CS100 >, < k, CS200 >\}$, *Offered_by*$=\emptyset >$, where i, j and k are all distinct. Let $t =< CS100, x >$ be a tuple on *Offered_by* and let us insert it into $\mathbf{r}$. By Theorem 1, to determine if $\mathbf{r} \cup \{t\}$ is consistent wrt F, input $\mathbf{r}$, t, and F into Algorithm 1 (shown in Section 2). Step (1) of Algorithm 1 generates a tuple $t'_1 =< CS100, i, CS >$ on {*Course*, *Instructor*, *Dept*}. The tuple q generated in step (2) is empty exactly when 'x' is not equal to 'CS'. So the updated state is consistent exactly when the *Dept*-component of the inserted tuple is 'CS'. □

4 Computing $[X]_{\mathbf{r}}$

In this section, we prove that ctm BCNF database schemes are bounded and, more interestingly, that total projections can be computed via unions of projections of extension joins.

Corollary 3: Let $\mathbf{R}$ be a ctm BCNF database scheme wrt F, where F is the set of key dependencies embedded in $\mathbf{R}$. Then $\mathbf{R}$ is bounded wrt F.

Proof: It follows from Theorems 5 and 4(b). □

The expressions to compute total projections given by Corollary 3 may be more complex than unions of projections of extension joins. However, the previously known subclasses of ctm BCNF database schemes have the property that the total projections of their representative instances can be computed with unions of projections of extension joins. This is also the case for the entire class of ctm BCNF database schemes.

Theorem 7: Let $\mathbf{R}$ be a ctm BCNF database scheme wrt F, where F is the set of key dependencies embedded in $\mathbf{R}$. Then for any $X \subseteq U$, and for any consistent state $\mathbf{r}$ of $\mathbf{R}$, $[X]_{\mathbf{r}}$, the X-total projection of the representative instance of $\mathbf{r}$, can be computed with a union of projections onto X of extension joins that cover X.

5 Conclusions

We have derived an efficient algorithm that recognizes exactly the class of BCNF database schemes which are constant-time-maintainable. We have also found a constant-time algorithm for solving the maintenance problem for this class of database schemes. Finally, we proved that this class of schemes is bounded and, more interestingly, that total projections can be computed via unions of projections of extension joins. This also showed that the class of schemes we identified is the largest known class of constant-time-maintainable schemes which is bounded.

Acknowledgements

E. Chan's work was supported by the Natural Sciences and Engineering Research Council of Canada; H. Hernández' work was supported by a grant from Texas A&M University. The authors thank an anonymous referee for some valuable comments on a previous version of this paper.

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.

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

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

[BB] Beeri, C., Bernstein, P.A. "Computational Problems Related to the Design of Normal Form Relational Schemas," *ACM TODS 4*, 1 (1979), pp. 30-59.

[BBG] Beeri, C., Bernstein, P.A., and Goodman, N. "A Sophisticate's Introduction to Database Normalization Theory," *Proc. VLDB 1978*, pp. 113-124.

[BG] Bernstein, P.A., Goodman, N. "What Does Boyce-Codd Normal Form Do?" *Proc. VLDB 1980*, pp. 245-259.

[BV] Brosda, V., Vossen, G., "Updating a Relational Database Through a Univer-

sal Schema Interface," *Proc. ACM PODS, 1985*, pp. 66-75.

[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., "Independence-reducible Database Schemes," to appear in *Proc. ACM PODS, 1988.*

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

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

[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.

[LeP] LeDoux, C.H., Parker, D.S. "Reflections on Boyce-Codd Normal Form," *Proc. VLDB 1982*, pp. 131-141.

[LO] Lucchesi, C.L., Osborn, S.L. "Candidate Keys for Relations," *Journal of Computer and System Sciences 17*, 2 (1978), pp. 270-279.

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

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

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

[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.

[O] Osborn, S.L. "Testing for Existence of a Covering Boyce-Codd Normal Form," *IPL 8*, 1 (1979), pp. 11-14.

[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.

[U] Ullman, J.D., *Principles of Database Systems*, Computer Science Press, Potomac, 1982.

[V] Vassiliou, Y., "A Formal Treatment of Imperfect Information in Data Management," *CSRG TR-123*, Nov. 1980, University of Toronto.

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