DATABASE DECOMPOSITION INTO FOURTH NORMAL FORM

Gösta Grahne and Kari-Jouko Räihä

University of Helsinki, Department of Computer Science
Tukholmankatu 2, SF-00250 Helsinki 25, Finland

Abstract. We present an algorithm that decomposes a database scheme when the dependency set contains functional and multivalued dependencies. The schemes in the resulting decomposition are in fourth normal form and have a lossless join. Our algorithm does not impose restrictions on the allowed set of dependencies, and it never requires the computation of the full closure of the dependency set. Furthermore, the algorithm works in polynomial time for classes of dependencies that properly contain conflict-free dependency sets.

1. INTRODUCTION

The structure of a relational database can be defined by a universal relation scheme U and by a set of dependencies D. Relations that conform to the universal scheme may contain much redundant information and have undesirable update properties [Dat81]. It is therefore customary to decompose the universal scheme into subschemes $R_1,\ldots,R_k$ such that $U = \bigcup_{i=1}^{k} R_i$. The collection $\{R_1,\ldots,R_k\}$ is called the database scheme. Universal relations can be represented as their projections onto the relation schemes in the database scheme.

This work was supported by the Academy of Finland.

A good decomposition should have several properties [BBG78]:

(1) *Minimal redundancy:* the relation schemes should represent meaningful objects so that, for instance, they can be updated without knowing the values for all the attributes in the universal scheme. This is usually taken to mean that the relation schemes should be in one of various normal forms.

(2) *Representation:* it should be possible to recover the universal relation from the component relations, i.e. the decomposition should have a lossless join.

(3) *Separation:* when a (projected) relation is updated, it should be possible to ensure that the dependencies in the universal relation still hold after the update without actually constructing the universal relation. This condition is satisfied if the decomposition is dependency preserving, i.e. if the projected dependencies imply the original set D.

There exists a wide variety of dependency types, each one giving rise to different sets of normal forms. The two most common dependency types are functional dependencies and multivalued dependencies. The normal forms associated with functional dependencies are third normal form (3NF) [Cod72] and Boyce-Codd normal form (BCNF) [Cod74], whereas fourth normal form (4NF) [Fag77] is defined for multivalued dependencies. The requirements for these normal forms increase from 3NF to 4NF, i.e. 4NF implies BCNF, which implies 3NF.

Unfortunately, it is not always possible to find a decomposition having all these desirable properties. In particular, there exist relation schemes (or rather dependencies) that do not have a dependency preserving decomposition into BCNF or 4NF [BeB79]. The question of what to do when a good decomposition does not exist is controversial (see e.g. [BBG78]). Especially the validity of the universal relation assumption has given rise to much debate (cf. [AtP82,Ull82a]).

It is, of course, important to assess how well each of the three properties meets its intended goals. But this is not enough. Even if it is possible to decompose a relation scheme into subschemes that satisfy some set of properties, the method will be of little use if the decomposition algorithm is prohibitively expensive.

In this paper we will focus our attention on a problem that is known to be always solvable: the problem of decomposing a scheme U into a set of subschemes $\{R_1,\ldots,R_k\}$ such that $R_1,\ldots,R_k$ have a lossless join and each R_i is in normal form.

The complexity of the problem depends heavily on the desired normal form. For 3NF a simple polynomial time synthesis algorithm was presented in [BDB79]. Finding a lossless decomposition into BCNF is more difficult, but an algorithm working in polynomial time was recently given by Tsou and Fischer [TsF80]. Their approach does not generalize to multivalued dependencies, and the complexity of lossless decomposition into 4NF is open. We shall give an algorithm that is more efficient than existing general algorithms, and that works in polynomial time for a broader class of dependencies than existing polynomial algorithms.

We use the notation and terminology of [Ull82b].

2. PREVIOUS WORK

There is a straightforward way of finding a lossless decomposition into 4NF. Suppose U has been replaced by a set of relation schemes $R_1,\ldots,R_k$ (initially k=1 and R_1=U). Suppose further that some R_i is not in 4NF. This means that there exists a dependency $X \twoheadrightarrow Y$ in D^+ whose projection onto R_i is nontrivial (i.e. $\emptyset \neq X \subset R_i$, $Y \cap X = \emptyset$, and $\emptyset \neq Y \cap R_i \neq R_i - X$) such that X is not a key of R_i. Then R_i is replaced by $XY \cap R_i$ and $X \cup (R_i - Y)$. This guarantees the losslessness of the join [Fag77,ABU79], and the repeated application produces a set of schemes in 4NF.

This algorithm has several efficiency problems. First, it is known that testing whether a relation scheme is in BCNF is NP-complete [BeB79]. Therefore testing the 4NF property is also computationally intractable. In their algorithm for functional dependencies [TsF80], Tsou and Fischer have circumvented this problem by testing only a sufficient BCNF condition, not a sufficient and necessary one. In other words, if a nontrivial dependency is found in R_i, then R_i is decomposed regardless of whether it actually is in BCNF or not. The same idea can be applied to multivalued dependencies and 4NF also. (If the set of dependencies includes only multivalued dependencies, the absence of nontrivial dependencies is a sufficient and necessary condition for 4NF.)

The problem then becomes that of finding a nontrivial dependency in some R_i. Fagin [Fag77] proposed to compute the dependency closure D^+ before running the decomposition algorithm. Then it would be sufficient just to scan through the precomputed set when searching for some nontrivial dependency $X \twoheadrightarrow Y$. Unfortunately, the size of the closure D^+ can be exponential [Ull82b], thereby ruining the efficiency of this approach. Moreover, Lien has shown in [Lie81] that if such a precomputed set of dependencies is used in the decomposition algorithm, then that set *must* be the closure D^+: if any smaller set is used, the algorithm may not work correctly.

Therefore the nontrivial dependency $X \twoheadrightarrow Y$ must be found differently. Another approach would be to avoid computing D^+ and to derive the dependencies on demand from the initial set D. Indeed, if there are any nontrivial dependencies in R_i, then at least one of them can be found efficiently using a result of [HIT79].

But then we are faced with the problem that the decomposition tree may grow too large. If we use an arbitrary nontrivial dependency $X \twoheadrightarrow Y$ for decomposing R_i, neither of the new schemes ($XY \cap R_i$ and $X \cup (R_i - Y)$) is necessarily in 4NF. Therefore both of them must be decomposed further. In each decomposition step the new subschemes must have less attributes than the decomposed scheme R_i, but the difference may be only one attribute. Thus the decomposition tree may have $|U|$ levels, and in the case of a full binary tree this means $2^{|U|}-1$ decomposition steps.

This indicates that instead of an arbitrary nontrivial dependency $X \twoheadrightarrow Y$, one should find a dependency that is in a sense as "small" as possible. That is, the subscheme $XY \cap R_i$ should not contain any nontrivial dependencies. This would guarantee that $XY \cap R_i$ is in 4NF, and the decomposition tree would degenerate into a linear tree with $O(|U|)$ nodes. In the case of functional dependencies, such a "smallest" $X \rightarrow Y$ can be found in polynomial time. Tsou and Fischer start with an arbitrary nontrivial dependency $X \rightarrow Y$. Then they repeatedly test whether some nontrivial dependency $X' \rightarrow Y'$ still holds in $XY \cap R_i$; if so, they throw out the attributes in $(XY \cap R_i) - X'Y'$. This is possible, since the properties of functional dependencies imply that $X' \rightarrow Y'$ also holds in R_i itself (and not just in $XY \cap R_i$). In this way the "smallest" nontrivial $X \rightarrow Y$ that holds in R_i is finally obtained, and it can be used to decompose R_i.

This approach does not work for multivalued dependencies. If we find some nontrivial $X' \twoheadrightarrow Y'$ in $XY \cap R_i$, it does *not* necessarily hold in R_i itself; i.e. $X' \twoheadrightarrow Y'$ may be embedded in $XY \cap R_i$. Finding efficiently the smallest nontrivial multivalued dependency $X \twoheadrightarrow Y$ appears to be a difficult problem, and it was left open by Tsou and Fischer. However, solving this problem is essential for making the decomposition efficient, if the dependencies are tested on the basis of D without computing D^+.

Our solution is in some sense between the two extremes. We avoid computing D^+, but we do not use the fixed set D, either. Instead, during the decomposition process we will add to D members of D^+ in such a way that whenever we wish to find a nontrivial dependency $X \twoheadrightarrow Y$, it is sufficient to examine only the dependencies in the extended dependency set D. This process is dynamic: it is guided by the decomposition tree. Therefore *all* the dependencies in D^+ will not be added to D.

There exist some algorithms for performing the decomposition efficiently, but none of these algorithms achieves all the desired properties. Lien's algorithm [Lie82] produces a lossless decomposition into 4NF schemes in polynomial time, but it only works for so-called conflict-free dependency sets. Sciore's method [Sci81] works under the same restriction. Although this is a practical subclass of multivalued dependencies, the general case (unrestricted dependencies) still remains as an intrigueging open question. Similarly, da Silva's algorithm [daS80] produces a lossless decomposition into schemes that are in 4NF as far as D is concerned; however, there may still exist derived dependencies in D^+ that violate the normal form conditions.

3. AN INFORMAL DESCRIPTION OF THE METHOD

We shall, for the sake of brevity, assume in the sequel that the dependency set D consists of only multivalued dependencies. If functional dependencies also are present we can either convert them into multivalued ones (by the axiom $X \rightarrow Y$ implies $X \twoheadrightarrow Y$), or start with Tsou and Fischer's algorithm to produce a lossless set of BCNF schemes, and then apply our algorithm to this set.

Our goal is to maintain a set of left-hand sides K, such that whenever there is a nontrivial dependency in a candidate scheme R, then there is at least one such dependency whose left-hand side is in K. Initially K equals the set of left-hand sides in D, denoted by LHS(D).

As an example, consider the set of dependencies given in [Lie81,p.61]. Here $U = ABCEGH$ and $D = \{A \twoheadrightarrow G \mid BHCE,\ \ B \twoheadrightarrow H \mid AGCE,\ \ GH \twoheadrightarrow C \mid E \mid AB\}$. (We have augmented the original D by listing the entire dependency basis on the right-hand side of each left-hand side of D.) If we let K equal LHS(D) permanently, then the resulting schemes may not be in 4NF. For instance, in the decomposition tree

$$\begin{tikzcd}
 & ABCEGH \arrow[dl, dash] \arrow[dr, dash] & & \\
AB & & ABHCE \arrow[dl, dash] \arrow[dr, dash] & \\
 & BH & & ABCE
\end{tikzcd}$$

which is obtained by using the dependencies $A \twoheadrightarrow G$ and $B \twoheadrightarrow H$, the scheme ABCE is not in 4NF: the dependencies $AB \twoheadrightarrow C$ and $AB \twoheadrightarrow E$ hold in it. Yet none of the original dependencies in D contradicts the 4NF property.

If the dependencies are applied in some other order, the result may in fact be a decomposition into 4NF schemes. However, Lien [Lie81] gives another example where 4NF schemes are not reached, no matter what order of dependencies is used in the decomposition.

The problem with the given decomposition tree is that when $A \twoheadrightarrow G$ is applied, the left-hand side GH is split. The corresponding dependencies cannot be directly applied in the subschemes, although the subschemes may contain some *derived* nontrivial dependencies, whose derivation requires the use of GH. A key idea in our algorithm is that we will in a sense perform one step of such a derivation simultaneously with the decomposition step.

For instance, although $GH \twoheadrightarrow C \mid E \mid AB$ cannot be applied in ABHCE, we can achieve the same effect by adding to K the left-hand side AH. That is, if a dependency $X \twoheadrightarrow Y$ is derived using GH, then we must first derive G. This requires the application of the dependency $A \twoheadrightarrow G$. By combining these two derivation steps we can directly use the dependencies $AH \twoheadrightarrow C \mid E \mid B$ in ABCHE.

Similarly, if ABCHE is decomposed using $B \twoheadrightarrow H$, the newly added left-hand side AH is split. Again, we can fix the situation by adding to K the left-hand side AB.

In general, if R is the scheme being decomposed using a dependency $X \twoheadrightarrow Y$, and if P is a left-hand side that is split in the decomposition, then the element that is added to K is $X \cup (P \cap Y)$. It can be shown that this is sufficient for achieving the desired effect: only left-hand sides in K need to be considered when searching for the nontrivial dependencies. We will return to the correctness issue in Section 5.

4. THE DECOMPOSITION ALGORITHM

We start with some necessary definitions. Let $U = \{A_1, \ldots, A_m\}$ be a relation scheme and $D = \{X_1 \twoheadrightarrow Y_1, \ldots, X_n \twoheadrightarrow Y_n\}$ be a set of multivalued dependencies. For any $X \subsetneq U$, the dependency basis of X (denoted by DEP(X)) is defined as a partition of U-X such that D implies $X \twoheadrightarrow Y$ if and only if Y is a union of some sets in DEP(X) (provided $X \cap Y = \emptyset$). The dependency basis always exists and it can be computed efficiently [Bee80,Gal82].

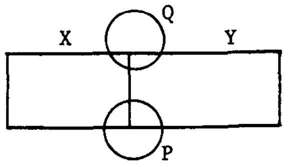
We say that a dependency $X \twoheadrightarrow Y$ is nontrivial in an attribute set R if $X \subsetneq X(Y \cap R) \subsetneq R$; otherwise $X \twoheadrightarrow Y$ is trivial in R (if $X \subseteq R$). For brevity, we will sometimes say that X is nontrivial in R if $X \twoheadrightarrow Y$ is nontrivial in R for some Y in DEP(X); otherwise X is said to be trivial in R.

As explained in Section 3, the splitting relation is essential to the decomposition algorithm. We say that $X \twoheadrightarrow Y$ splits an attribute set P if the following conditions are satisfied:

(i) $\emptyset \neq Y \cap P \neq P$, and

(ii) for some $Q \in DEP(P)$, $\emptyset \neq Y \cap Q \neq Y - P$.

Pictorially the splitting situation is the following.



A slightly stronger form of condition (i), sometimes referred to as the split key property, is discussed in e.g. [Lie82,BFM81,BeK83]. Here we are interested in the splitting of P only if P could be used to further decompose XY. Therefore we require also condition (ii), which guarantees the existence of a suitable Q in DEP(P). The algorithm would work correctly without condition (ii), but in this way it is more efficient.

We are now ready to present the decomposition algorithm.

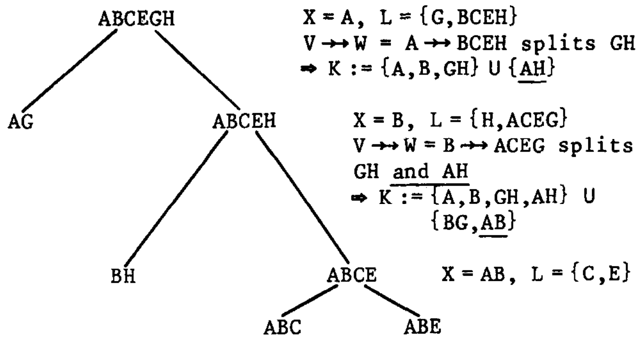
Algorithm: Decomposition of a relation scheme.

Input: A relation scheme U and a set of dependencies $D=\{X_1 \twoheadrightarrow Y_1,\dots,X_n \twoheadrightarrow Y_n\}$ such that $Y_i \in \mathrm{DEP}(X_i)$, $1 \leq i \leq n$.

Output: A decomposition $\pi=\{R_1,\dots,R_m\}$ of U such that $\bigcup_{i=1}^{m} R_i = U$, the decomposition has a lossless join, and each R_i is in 4NF.

```
Method:
K := LHS(D); π := {U};   {initialization}
{decomposition loop}
while there exist $R \in \pi$, $X \in K$ and $Y \in \mathrm{DEP}(X)$ such
      that $X \twoheadrightarrow Y$ is nontrivial in R
do begin
    let $L = \{Y \mid Y \in \mathrm{DEP}(X)$ and $Y \cap R \neq \emptyset\}$;
    {prepare for the decomposition using X: add
    to K new left sides}
    while there exist $Y \in L$, $V \in K$, $P \in K$ and $W \in$
          $\mathrm{DEP}(V)$ such that $V \subseteq XY \cap R \subseteq VW$
          {$V \twoheadrightarrow W$ is trivial in $XY \cap R$}
          and $V \twoheadrightarrow W$ splits P and $V(P \cap W) \notin K$
    do $K := K \cup \{V(P \cap W)\}$;
    $\pi := (\pi - \{R\}) \cup \bigcup_{Y \in L} \{X(Y \cap R)\}$;   {decomposition}
end; □
```

The following figure illustrates how the algorithm could work when it is applied to U = ABCEGH and $D=\{A \twoheadrightarrow G \mid BCEH,\ B \twoheadrightarrow H \mid ACEG,\ GH \twoheadrightarrow C \mid E \mid AB\}$.



ABCEGH
$X=A,\ L=\{G,BCEH\}$
$V \twoheadrightarrow W = A \twoheadrightarrow BCEH$ splits GH
$\Rightarrow K := \{A,B,GH\} \cup \{\underline{AH}\}$
AG
ABCEH
$X=B,\ L=\{H,ACEG\}$
$V \twoheadrightarrow W = B \twoheadrightarrow ACEG$ splits $\underline{\text{GH and AH}}$
$\Rightarrow \underline{K := \{A,B,GH,AH\}} \cup \{BG,\underline{AB}\}$
BH
ABCE
$X=AB,\ L=\{C,E\}$
ABC
ABE

The underlined parts indicate the essential facts that allow the algorithm to decompose the unnormalized ABCE further, even though all the members of LHS(D) are trivial in it.

Note also that the algorithm does not fix any particular order for considering the elements of K in the decomposition process. A different choice of the X-sets would yield a different decomposition tree.

5. CORRECTNESS OF THE DECOMPOSITION ALGORITHM

We will show that the algorithm works correctly by using an indirect proof. If $S \twoheadrightarrow T$ is nontrivial in some R_i belonging to the final decomposition, we will claim that S is necessarily added to K in the algorithm. Therefore R_i could have been decomposed further using S, contradicting the fact that R_i belongs to the final output of the algorithm.

To prove that S is added to K it is necessary to study the properties of splittings. The first two lemmas prove two essential facts used in the correctness proof. In both lemmas we assume that $R \subseteq U$ and that $S \twoheadrightarrow T$ is nontrivial in R. Furthermore, we assume that S is minimal among such left sides, i.e. if S' is nontrivial in R then $|S| \leq |S'|$.

It is useful to consider how S is related to LHS(D). We say that the pair

$(\{S_1,\ldots,S_k\},\{S_{k+1},\ldots,S_{k+t}\})$ is a representation of S if the following conditions are satisfied:

(i) $S_i \in \mathrm{LHS}(D)$, $1 \le i \le k+t$.

(ii) $\bigcup_{i=1}^{k} S_i = \bigcup \{X \mid X \in \mathrm{LHS}(D) \text{ and } X \subseteq S\}$.

(iii) $\emptyset \ne S_i \cap S \ne S_i$, $k+1 \le i \le k+t$.

(iv) $S = (\bigcup_{i=1}^{k} S_i) \cup (\bigcup_{i=k+1}^{k+t} (S_i \cap S))$.

Furthermore, we say that $(\{S_1,\ldots,S_k\},\{S_{k+1},\ldots,S_{k+t}\})$ is a minimal representation of S (denoted by $M(S)$) if for any other representation $(\{T_1,\ldots,T_{k'}\},\{T_{k'+1},\ldots,T_{k'+t'}\})$ we have $k+t \le k'+t'$. Intuitively, a minimal representation of S is a minimal cover of S using sets in $\mathrm{LHS}(D)$, subject to the condition that if $X \subseteq S$ and $X \in \mathrm{LHS}(D)$, then X is covered by sets that are completely contained in S. If $X \twoheadrightarrow Y$ is a dependency in D^+ where X is nonredundant, then a minimal representation for X always exists, but it need not be unique.

In the lemmas we assume that $M(S) = (\{S_1,\ldots,S_k\},\{S_{k+1},\ldots,S_{k+t}\})$ and that $k+t \ge 2$. From the minimality of S it follows that each S_i, $1 \le i \le k$, is trivial in R; let $T_i \in \mathrm{DEP}(S_i)$ such that $R \subseteq S_i T_i$.

As a notational convenience, for any collection of attribute sets $\{X_1,\ldots,X_n\}$ where each $X_i \subseteq U$, we define $\bigcup_{i\in\emptyset} X_i = \emptyset$ and $\bigcap_{i\in\emptyset} X_i = U$.

Lemma 1. There exists a $P \in \mathrm{LHS}(D)$ such that for any $I \subsetneq \{1,\ldots,k\}$ and $j \in \{1,\ldots,k\}-I$, the following hold:

(i) $S_j \twoheadrightarrow T_j$ splits $(\bigcup_{i\in I} S_i) \cup (P \cap \bigcap_{i\in I} T_i)$, and

(ii) $P \cap \bigcap_{i=1}^{k} T_i \subseteq \bigcup_{i=k+1}^{k+t} (S_i \cap S)$.

Proof. Consider the computation of $\mathrm{DEP}(S)$ using Beeri's algorithm [Bee80]. The algorithm starts with a candidate basis, e.g. $\{U-S\}$, but any partition that is not finer than $\mathrm{DEP}(S)$ could be used. Then the basis is refined using the following rule: If T is in the partition and $X \twoheadrightarrow Y$ is in D such that $X \cap T = \emptyset$, then T is replaced by $T \cap Y$ and $T-Y$. The algorithm terminates when the partition cannot be further refined.

By the decomposition rule for multivalued dependencies, we may in this case take $\{(\bigcap_{i=1}^{k} T_i)-S,\ U-S-\bigcap_{i=1}^{k} T_i\}$ as the candidate basis. Now $R \subseteq \bigcap_{i=1}^{k} S_i T_i \subseteq \bigcap_{i=1}^{k} ST_i = SU(\bigcap_{i=1}^{k} T_i) = S \cup ((\bigcap_{i=1}^{k} T_i)-S)$; thus $S \twoheadrightarrow (\bigcap_{i=1}^{k} T_i)-S$ is trivial in R. Since we assumed that S is nontrivial in R, there must exist a $P \twoheadrightarrow Q$ in D such that

$$P \cap ((\bigcap_{i=1}^{k} T_i)-S) = \emptyset \tag{1}$$

and $\emptyset \ne Q \cap ((\bigcap_{i=1}^{k} T_i)-S) \ne (\bigcap_{i=1}^{k} T_i)-S$. Therefore

$$\emptyset \ne Q \cap ((\bigcap_{i\in I} T_i)-S) \ne (\bigcap_{i\in I} T_i)-S \tag{2}$$

for any $I \subseteq \{1,\ldots,k\}$.

On the other hand, the minimality of S and $M(S)$ implies that $\bigcup_{i\in I} S_i$ is trivial in R for any $I \subseteq \{1,\ldots,k\}$ such that $I \ne \{1,\ldots,k+t\}$. Consider the computation of $\mathrm{DEP}(\bigcup_{i\in I} S_i)$. A candidate basis is $\{\bigcap_{i\in I} T_i, U-(\bigcup_{i\in I} S_i)-(\bigcap_{i\in I} T_i)\}$. Since we just have shown that Q would partition $\bigcap_{i\in I} T_i$, $P \twoheadrightarrow Q$ cannot be used to refine $\bigcap_{i\in I} T_i$. Therefore

$$P \cap (\bigcap_{i\in I} T_i) \ne \emptyset. \tag{3}$$

We will show that P has the properties claimed in the lemma. Let I be an arbitrary proper subset of $\{1,\ldots,k\}$, and let j be an arbitrary element of $\{1,\ldots,k\}-I$. We will denote by V the set $(\bigcup_{i\in I} S_i) \cup (P \cap \bigcap_{i\in I} T_i)$. To show that $S_j \twoheadrightarrow T_j$ splits V, two conditions must be satisfied:

(a) $\emptyset \ne T_j \cap V \ne V$, and

(b) for some $W \in \mathrm{DEP}(V)$, $\emptyset \ne T_j \cap W \ne T_j - V$.

We will deal with the four inequalities one at a time.

1° $\emptyset \neq T_j \cap V$. If $I \cup \{j\} \neq \{1,\dots,k+t\}$, (3) implies that $\emptyset \neq P \cap (\bigcap_{i \in I \cup \{j\}} T_i) = T_j \cap (P \cap \bigcap_{i \in I} T_i)$, which immediately gives the result. If $I \cup \{j\} = \{1,\dots,k+t\}$, then $t=0$, $k \geq 2$, and $I \neq \emptyset$; let $h \in I$. By the minimality of S and M(S), $S_h - S_j \neq \emptyset$; and because $S_h \subseteq R \subseteq S_j T_j$, $S_h \cap T_j \neq \emptyset$. The result follows.

2° $T_j \cap V \neq V$. Suppose to the contrary that $T_j \cap V = V$, i.e. $V \subseteq T_j$. In particular, $P \cap \bigcap_{i \in I} T_i \subseteq T_j$. Let $H = \{1,\dots,k\}-\{j\}$; thus $\bigcap_{i=1}^{k} T_i = \bigcap_{i \in H} T_i$. But (1) implies that $P \cap (\bigcap_{i=1}^{k} T_i) = \emptyset$, while (3) implies that $P \cap (\bigcap_{i \in H} T_i) \neq \emptyset$: a contradiction.

3° For some W in DEP(V), $T_j \cap W \neq \emptyset$. To find a suitable W, we will start from the dependency $\bigcup_{i \in I} S_i \twoheadrightarrow \bigcap_{i \in I} T_i$. By shifting those attributes of $\bigcap_{i \in I} T_i$ that belong to P to the left, we get

$V = (\bigcup_{i \in I} S_i) \cup (P \cap (\bigcap_{i \in I} T_i)) \twoheadrightarrow$

$(\bigcap_{i \in I} T_i) - (P \cap (\bigcap_{i \in I} T_i)) = (\bigcap_{i \in I} T_i) - P.$

Since $((\bigcap_{i \in I} T_i) - P) \cap P = \emptyset$, $P \twoheadrightarrow Q$ can be used to refine $(\bigcap_{i \in I} T_i) - P$. Thus $V \twoheadrightarrow ((\bigcap_{i \in I} T_i) - P) \cap Q = (\bigcap_{i \in I} T_i) \cap Q - (P \cap Q) = (\bigcap_{i \in I} T_i) \cap Q$, since $Q \in$ DEP(P). We will show that condition (b) holds for $(\bigcap_{i \in I} T_i) \cap Q$; then it obviously holds for some W in DEP(V) such that $W \subseteq (\bigcap_{i \in I} T_i) \cap Q$. The first part, $T_j \cap (\bigcap_{i \in I} T_i) \cap Q \neq \emptyset$, follows immediately from (2).

4° $T_j \cap (\bigcap_{i \in I} T_i) \cap Q \neq T_j - V$. From (2) we know that $Q \cap ((\bigcap_{i=1}^{k} T_i) - S) \neq (\bigcap_{i=1}^{k} T_i) - S$, which implies $(\bigcap_{i=1}^{k} T_i) - S - Q \neq \emptyset$. Let $A \in (\bigcap_{i=1}^{k} T_i) - S - Q$. Since $A \notin Q$, $A \notin T_j \cap (\bigcap_{i \in I} T_i) \cap Q$. We will establish the claim by showing that $A \in T_j - V$. Since $A \in \bigcap_{i=1}^{k} T_i$, we immediately have $A \in T_j$. Moreover, $A \notin S$ implies $A \notin \bigcup_{i \in I} S_i$, and $A \in (\bigcap_{i=1}^{k} T_i) - S$ implies $A \notin P$ (using (1)). Thus $A \notin (\bigcup_{i \in I} S_i) \cup (P \cap (\bigcap_{i \in I} T_i)) = V$.

We have shown that part (i) of the claim holds. Part (ii) follows directly from (1): since $P \cap ((\bigcap_{i=1}^{k} T_i) - S) = P \cap ((\bigcap_{i=1}^{k} T_i) - ((\bigcup_{i=1}^{k} S_i) \cup (\bigcup_{i=k+1}^{k+t} (S_i \cap S))))$ $= P \cap ((\bigcap_{i=1}^{k} T_i) - (\bigcup_{i=k+1}^{k+t} (S_i \cap S))) = \emptyset$, we have $P \cap (\bigcap_{i=1}^{k} T_i) \subseteq \bigcup_{i=k+1}^{k+t} (S_i \cap S)$. □

<u>Lemma 2</u>. Let V be an attribute set such that $\bigcup_{i=1}^{k} S_i \subseteq V \subsetneq S$. Then there exist $W \in$ DEP(V) and $P \in$ LHS(D) such that $V \twoheadrightarrow W$ splits P and $P \cap W \subseteq S$.

<u>Proof</u>. By the minimality of S, $V \twoheadrightarrow W$ is trivial in R for any $W \in$ DEP(V). Let W be the unique set in DEP(V) such that $R \subseteq VW$.

Consider the computation of DEP(S). A candidate basis is $\{W-S, U-W-S\}$. Since $S \twoheadrightarrow T$ is nontrivial in R and since $R \subseteq VW \subseteq SW = S \cup (W-S)$, there must exist a $P \twoheadrightarrow Q$ in D such that $P \cap (W-S) = \emptyset$ and $\emptyset \neq Q \cap (W-S) \neq W-S$. Thus clearly $P \cap W \subseteq S$; we claim that $V \twoheadrightarrow W$ splits P.

1° $P \cap W \neq \emptyset$. If $P \cap W = \emptyset$, $P \twoheadrightarrow Q$ could be used in the computation of DEP(V) to refine V, a contradiction.

2° $P \cap W \neq P$. If $P \cap W = P$, we would have $P \subseteq W$ and $P \subseteq S$. By the definition of M(S), $P \subseteq \bigcup\{X \mid X \in \text{LHS(D) and } X \subseteq S\} = \bigcup_{i=1}^{k} S_i \subseteq V$, contradicting the fact that $V \cap W = \emptyset$.

3° $Q \cap W \neq \emptyset$. Obvious.

4° $Q \cap W \neq W-P$. Since $Q \cap (W-S) \neq W-S$, we have $W-S-Q \neq \emptyset$. Let $A \in W-S-Q$; then $A \notin Q \cap W$. Moreover, $A \in W$, but $A \notin P$ because $P \cap (W-S) = \emptyset$, implying $A \in W-P$. □

<u>Theorem 1</u>. Consider $\pi = \{R_1,\dots,R_m\}$ and K at the beginning of the outer <u>while</u>-loop. Let S be a minimal set such that $S \twoheadrightarrow T$ is nontrivial in some R_i for some $T \in$ DEP(S). Then $S \in K$.

Proof. Let $M(S) = (\{S_1,\ldots,S_k\},\{S_{k+1},\ldots,S_{k+t}\})$. By the initialization of K, $S_i \in K$ for $1 \leq i \leq k+t$. If $k+t = 1$, then clearly $k = 1$ and $t = 0$, and $S = S_1 \in K$.

Consider then the case $k+t \geq 2$. The conditions for Lemma 1 are now satisfied. Let P be the set in LHS(D) whose existence is implied by Lemma 1.

Taking $I = \emptyset$ and $j = 1$, Lemma 1 (i) gives that $S_1 \twoheadrightarrow T_1$ splits P. By the minimality of S, S_1 is trivial in R_i. Therefore $S_1 \subseteq R_i = XY \cap R \subseteq S_1T_1$ during the decomposition step that produced R_i from R using some dependency $X \twoheadrightarrow Y$. Since $S_1 \twoheadrightarrow T_1$ splits P, $S_1 \cup (P \cap T_1)$ was added to K (unless it was there already).

By Lemma 1 (i), $S_2 \twoheadrightarrow T_2$ splits $S_1 \cup (P \cap T_1)$. Repeating the above argument yields that $S_2 \cup ((S_1 \cup (P \cap T_1)) \cap T_2) = S_2 \cup (S_1 \cap T_2) \cup (P \cap T_1 \cap T_2) = \bigcup_{i=1}^{2} S_i \cup (P \cap \bigcap_{i=1}^{2} T_i)$ is added to K. Straightforward inductive application of Lemma 1 (i) implies that $V = \bigcup_{i=1}^{k} S_i \cup (P \cap \bigcap_{i=1}^{k} T_i)$ is added to K during decomposition using $X \twoheadrightarrow Y$. Moreover, Lemma 1 (ii) states $P \cap \bigcap_{i=1}^{k} T_i \subseteq \bigcup_{i=k+1}^{k+t} (S_i \cap S)$; thus $V \subseteq S$.

If $V = S$, the claim follows. If $V \subsetneq S$, Lemma 2 implies that there exist $W \in DEP(V)$ and $P' \in LHS(D)$ such that $V \twoheadrightarrow W$ splits P' and $P' \cap W \subseteq S$. By the definition of splitting, $P' \cap W \neq \emptyset$, which together with the minimality of S implies that $R_i \subseteq VW$. But then $V \cup (P' \cap W)$ is added to K during the same decomposition step.

We clearly have $V \subsetneq V \cup (P' \cap W) \subseteq S$. If $V \cup (P' \cap W) \subsetneq S$, the above argument and Lemma 2 can be applied inductively, until we end up with S being added to K. □

Corollary. The decomposition algorithm terminates correctly.

Proof. Termination follows from the finiteness of U and D and from the fact that only nontrivial dependencies are used in the decomposition. The lossless join property is obvious from the results in [Fag77,ABU79]. To see that the resulting schemes are in 4NF, suppose to the contrary that for some R_i belonging to the final decomposition there exists an $S \subsetneq U$ such that S is nontrivial in R_i. Without loss of generality, let S be minimal among such left sides. By Theorem 1, $S \in K$, contradicting the fact that R_i belongs to the final decomposition even though it could be decomposed further using S. □

6. EFFICIENCY OF THE DECOMPOSITION ALGORITHM

Besides the actual decomposition π our algorithm computes the set K, which is a subset of $LHS(D^+)$. Therefore the algorithm is more efficient than the straightforward method where the entire D^+ is computed. Unfortunately, in the general case we have been unable to determine exactly how much smaller K is when compared to $LHS(D^+)$. There exist pathological examples where it appears difficult to bound the size of K by a polynomial. Thus the complexity of the algorithm is still open.

Before looking at the difficult cases, let us examine how the basic algorithm in Section 4 could be improved. There are two main approaches that can be used. First, the set K computed by the algorithm is still too large: there exist sets in K that do not serve any useful purpose. Thus we could place additional requirements for a set to be added to K, and prove that the algorithm still works correctly.

We have actually already met one such requirement: condition (ii) in the definition of splitting was introduced for the purpose of restricting the size of K. Another requirement can be added by examining the proof of Theorem 1. We can observe that whenever the proof refers to an element in K-LHS(D), the element is of the form $V \cup (W \cap P)$ where either $V \in LHS(D)$ or $P \in LHS(D)$. Thus we can immediately add the condition

$V \in LHS(D)$ or $P \in LHS(D)$

to the inner <u>while</u>-loop of the algorithm without violating its correctness.

Another way to enhance the efficiency of the decomposition algorithm is to choose a decomposition order that keeps both π and K small. Recall that the algorithm works correctly independently of the order in which the sets in K are considered; thus we are free to choose a convenient order as we please.

We will require that the sets are chosen from K in an order that is compatible with the following <u>containment order</u>. Let $X \in K$, $X' \in K$, $X \subsetneq X'$, so that both X and X' are nontrivial in some $R \in \pi$; then X' is *not* chosen before X as the left-hand side to be used in the decomposition. The same order is used by Lien [Lie82], who calls it a "p-ordering".

Intuitively, the motivation for using the containment order is that if X is used for decomposing R, then X' is contained in at most one of the resulting schemes. In the opposite case X would be contained in *all* the sons of R and could possibly be used to decompose several of them further. The following theorem establishes this property formally.

<u>Theorem 2</u>. Consider the sets π and K at the beginning of the outer <u>while</u>-loop. Let $Z \subsetneq R \in \pi$ such that Z is nontrivial in R. For all $T \in \pi - \{R\}$, $Z \not\subseteq T$.

<u>Proof</u>. By induction on $|\pi|$.

Basis. $\pi = \{U\}$. Obvious.

Inductive step. Suppose that the claim holds for K and π, which are replaced by K' and $\pi' = (\pi - \{R\}) \cup \bigcup_{i=1}^{p} \{X(Y_i \cap R)\}$ during a single execution of the outer <u>while</u>-loop. We will show that the claim holds for K' and π'. Let $Z \subsetneq R' \in \pi'$ such that Z is nontrivial in R'.

Consider first the case $R' \in \pi$. By the inductive hypothesis, $Z \not\subseteq T$ for any $T \in \pi - \{R'\}$. In particular, $Z \not\subseteq R$, which yields $Z \not\subseteq X(Y_i \cap R)$ for all i. This proves the claim.

Suppose then that $R' \notin \pi$, i.e. $R' = X(Y_i \cap R)$ for some i. Clearly Z is nontrivial in R; by the inductive hypothesis, $Z \not\subseteq T$ for any $T \in \pi - \{R\}$. Suppose (contrary to the claim) that $Z \subseteq X(Y_j \cap R)$ for some $j \neq i$. But then $Z \subsetneq X$. By Theorem 1, there exists a minimal $S \in K$ such that S is nontrivial in R. By the minimality of S, $S \subseteq Z \subsetneq X$. This contradicts the containment order of decomposition and proves the claim. □

Theorem 2 shows that each set in K can be used in the decomposition at most once. We also know that each basic step of the algorithm can be carried out efficiently: the main operation required is the computation of a dependency basis which can be done in almost linear time [Gal82]. Thus our algorithm works in polynomial time exactly when $|K|$ is bounded by a polynomial of $|D|$ and $|U|$.

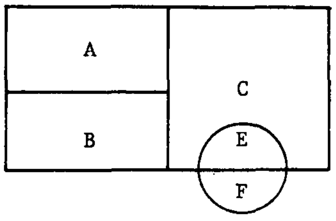
For restricted classes of dependencies it is possible to show that K does not grow excessively. So-called conflict-free sets of dependencies are the largest class that can be decomposed efficiently using previous algorithms (e.g. Lien's algorithm [Lie81,Lie82]). A conflict-free set is characterized in [BFM81] as a set that (i) does not split keys (where a "key" means a left-hand side of a dependency), and (ii) satisfies one of several equivalent properties, such as the orthogonality property or the intersection property.

Clearly, requirement (ii) is immaterial from the point of view of our algorithm. As long as condition (i) is satisfied, no sets are added to K, and the algorithm terminates in polynomial time. This trivially defines a class of dependencies properly containing conflict-free sets.

The really interesting cases are, of course, those where there exist splittings among members of LHS(D) (and thus K). If the splitting relation is acyclic, it is not difficult to find a decomposition order (a refinement of the containment order) that guarantees the efficiency of the algorithm. This still holds true for dependency sets containing cyclic splittings in a restricted, regular manner. A definition of such a class would be technical, with no apparent intuitive interpretation, so we refrain from giving one.

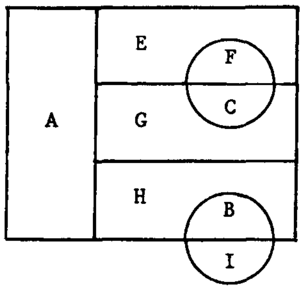
As mentioned before, there are cases where it appears difficult to derive a polynomial upper bound for $|K|$. We conclude this section with two simple examples that can be combined and expanded to produce complicated splittings.

It may not be immediately clear why the inner loop is required in the decomposition algorithm, i.e. why do we in a sense perform a group of "trivial decompositions" simultaneously with the "real" decomposition based on X. The following example illustrates this situation. Here $U = ABCEF$ and $D = \{AB \twoheadrightarrow CE \mid F,\ C \twoheadrightarrow ABE \mid F,\ EF \twoheadrightarrow A \mid BC\}$; U is illustrated by the following diagram.



Suppose that AB is used first to decompose U, resulting in the schemes ABCE and ABF. Now AB does not split either C or EF; although EF breaks into two pieces, it does not have a right-hand side that could partition $ABCE-AB-EF = C$. Therefore no sets are added to K because of AB. However, C becomes trivial during the same decomposition step, and $C \twoheadrightarrow ABE$ *does* split EF. Therefore $C \cup (ABE \cap EF) = CE$ is added to K, and we can in the next step use $CE \twoheadrightarrow A \mid B \mid F$ to further decompose ABCE into ACE and BCE.

In this case the new set is immediately nontrivial in some scheme. The next example illustrates a situation where we reach a nontrivial set only through a chain of trivial sets. Let $U = ABCEFGHI$ and $D = \{A \twoheadrightarrow BCEFGH \mid I,\ BGH \twoheadrightarrow AC \mid E \mid F \mid I,\ BI \twoheadrightarrow ACGH \mid E \mid F,\ CF \twoheadrightarrow BH \mid AEGI\}$ (see the following diagram).



Suppose now that decomposition begins by using BGH, i.e. $\{U\}$ is replaced by $\{ABCGH, BEGH, BFGH, BGHI\}$. Again, BGH does not split any of the other left-hand sides, but since also A becomes trivial in ABCGH and since A splits both BI and CF, we add AB and AC to K. Computing the dependency bases we find that $AB \twoheadrightarrow CGH \mid E \mid F \mid I$ and $AC \twoheadrightarrow BEFGH \mid I$ are both trivial in ABCGH; therefore they are also considered in the role of V within the inner loop of the algorithm. In particular, since AB splits CF, the set ABC is added to K. Now $ABC \twoheadrightarrow G \mid H \mid E \mid F \mid I$, so ABC can be used to further decompose ABCGH.

These examples show that, as regards the inner loop of the algorithm, there may be many sets in the role of V, and also that it is necessary to consider the newly added trivial sets in the role of V. By generalizing these examples so that they involve many alternatives instead of only two, it is possible to create situations where K grows excessively during a decomposition step.

As stated before, all the sets that are added to K are not useful. The problem of restricting the growth of K, either by adding new requirements into the algorithm or by refining the decomposition order, is left for further study.

7. CONCLUDING REMARKS

We have given an algorithm that decomposes a universal relation scheme into a set of schemes that are in 4NF and have a lossless join. The algorithm works for any set of functional and multivalued dependencies. The key idea was to apply the concept of splittings among the attribute sets in order to avoid the computation of the entire closure of the dependency set. This is in contrast with most of the previous methods, which downright forbid splittings (e.g. [Lie82]).

The algorithm works in polynomial time for classes of dependencies that properly contain conflict-free dependency sets. It is an open question whether improvements in the basic algorithm or in the decomposition order can make the algorithm run in polynomial time in the general case.

REFERENCES

ABU79 A.V. Aho, C. Beeri & J.D. Ullman, The theory of joins in relational databases. ACM Transactions on Database Systems 4,3 (Sept. 1979), 297-314.

AtP82 P. Atzeni & D.S. Parker, Assumptions in relational database theory. Proc. of the ACM Symposium on Principles of Database Systems, March 1982, 1-9.

BBG78 C. Beeri, P.A. Bernstein & N. Goodman, A sophisticate's introduction to database normalisation theory. Proc. of the Fourth International Conference on Very Large Data Bases, Sept. 1978, 113-124.

BeB79 C. Beeri & P.A. Bernstein, Computational problems related to the design of normal form relation schemes. ACM Transactions on Database Systems 4,1 (March 1979), 30-59.

Bee80 C. Beeri, On the membership problem for functional and multivalued dependencies in relational databases. ACM Transactions on Database Systems 5,3 (Sept. 1980), 241-259.

BFM81 C. Beeri, R. Fagin, D. Maier & M. Yannakakis, On the desirability of acyclic database schemes. Research Report RJ3131, IBM, San Jose, May 1981.

BeK83 C. Beeri & M. Kifer, Elimination of intersection anomalies from database schemes. Proc. of the Second ACM Symposium on Principles of Database Systems, March 1983, 340-351.

BDB79 J. Biskup, U. Dayal & P.A. Bernstein, Synthesizing independent database schemas. Proc. ACM SIGMOD 1979 Conference on Management of Data, 1979, 143-152.

Cod72 E.F. Codd, Further normalization of the data base relational model. Data Base Systems, R. Rustin (ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, 33-64.

Cod74 E.F.Codd, Recent investigations in relational database systems. Information Processing 74, Proc. of IFIP Congress 74, J.L. Rosenfeld (ed.), North-Holland Publ. Co., Amsterdam, 1974, 1017-1021.

daS80 A.C. da Silva, The decomposition of relations based on relational dependencies. Ph. D. thesis, Univ. of California, Los Angeles, 1980.

Dat81 C.J. Date, An Introduction to Database Systems. Addison-Wesley, Reading, Mass., 1981.

Fag77 R. Fagin, Multivalued dependencies and a new normal form for relational databases. ACM Transactions on Database Systems 2,3 (Sept. 1977), 534-544.

Gal82 Z. Galil, An almost linear-time algorithm for computing a dependency basis in a relational database. J. ACM 29,1 (Jan. 1982), 96-102.

HIT79 K. Hagihara, M. Ito, K. Taniguchi & T. Kasami, Decision problems for multivalued dependencies in relational databases. SIAM Journal on Computing 8,2 (May 1979), 247-264.

Lie81 Y.E. Lien, Hierarchical schemata for relational databases. ACM Transactions on Database Systems 6,1 (March 1981), 48-69.

Lie82 Y.E. Lien, On the equivalence of database models. J. ACM 29,2 (April 1982), 333-362.

Sci81 E. Sciore, Real-world MVD's. Proc. ACM SIGMOD 1981 Conference on Management of Data, April 1981, 121-132.

TsF80 D.-M. Tsou & P.C. Fischer, Decomposition of a relation scheme into Boyce-Codd normal form. Proc. ACM 1980 Annual Conference, 1980, 411-417. Reprinted in ACM SIGACT News 14,3 (1982), 23-29.

Ull82a J.D. Ullman, The U.R. strikes back. Proc. of the ACM Symposium on Principles of Database Systems, March 1982, 10-22.

Ull82b J.D. Ullman, Principles of Database Systems (Second Edition). Computer Science Press, Potomac, Md., 1982.