*Appl. Math. and Comp. Sci., 1996, vol.6, No.1, 137–157*

# COMPUTING A COVER FOR PROJECTED FUNCTIONAL DEPENDENCIES FROM A BOOLEAN EXPRESSION

WOJCIECH ZAWADZKI*

This paper gives the solution to a problem of finding an expression for a cover for $\pi_R(F)$, where $F$ is a set of functional dependencies, using Boolean functions as a formal tool. Another approach to represent a set of functional dependencies $F$ by a Boolean function $\varphi(R)$, where $R = \{A, B, \ldots\}$ stands for a relation schema, is presented. The main result is an algorithm of transformation: $\varphi(R) \rightarrow \varphi(\pi_X(R)) \rightarrow \pi_X(F)$. It is shown that such a transformation is equivalent to the decomposition of a Boolean function. The algorithm employs a number of optimization steps to reduce its complexity and to avoid redundancies resulting from the augmentation rule. A paper is being prepared (Zawadzki, 1995) in which the algorithm will be implemented and some estimation of time complexity will be given. It is conjectured that it may run in polynomial time unless the number of non-redundant dependencies is itself an exponential function of $|X|$. To the author's knowledge, there is only one algorithm (Gottlob, 1987) for the above problem, but it is not guaranteed to run in polynomial time.

## 1. Introduction

A *relation scheme* $R$ is a finite set $\{A_1, A_2, \ldots, A_n\}$ of symbols called *attributes* such that each attribute is associated with a domain $\mathrm{dom}(A_i)$ which is the set of all possible values for the respective attribute. We shall use the letters $A$, $B$, $C$ in order to refer to individual attributes and the letters $X$, $Y$, $Z$, $V$ in order to refer to sets of attributes. The union of $X$ and $Y$ will be denoted by $XY$.

A *relation* (also called *instance*) on a relation scheme $R(A_1, A_2, \ldots, A_n)$ is a subset of the Cartesian product $\mathrm{dom}(A_1) \times \mathrm{dom}(A_2) \times \cdots \times \mathrm{dom}(A_n)$; relations will be denoted by $r_1, r_2, \cdots$. The elements of a relation are called *tuples* and denoted by $t_1, t_2, \cdots$. If $t$ is a tuple on $R$, $A \in R$, then $t[A]$ will denote the value of $t$ with regard to $A$. Similarly, $t[X]$ will denote the sequence of values $t[A_1]$, $t[A_2], \ldots$, where $A_1, A_2, \ldots \in X$, $\mathrm{dom}(X) = \mathrm{dom}(A_1) \times \mathrm{dom}(A_2) \times \cdots$.

Let $X, Y \subseteq R$. A *functional dependency* (FD) $X \rightarrow Y$ is satisfied by a relation $r$ over $R$ iff for any tuple $t_1$, $t_2$, whenever $t_1[X] = t_2[X]$, we have $t_1[Y] = t_2[Y]$. A functional dependency holds in $R$ iff it is satisfied by every relation over $r$. A set

* Kuwait University, Department of Mathematics and Computer Science, P.O. Box 5969, Safat, 13060 Kuwait

of all FD's for $R$ will be denoted by $F$. The *closure* of $F$, denoted by $F^+$, is the set of all FD's that are logically implied by $F$. The *cover* of $F$, denoted by $F_C$, is the set of FD's such that $F$ logically implies all dependencies in $F_C$ and vice versa.

If $R$ is a relation scheme and $\rho(R_1, \ldots, R_k)$ is a decomposition, then $\pi_{R_i}(r)$ is a *projection* of some $r$ on $R$ onto $R_i$. The projection of $F$ onto a set of attributes $Z$, denoted by $\pi_Z(F)$, is the set of dependencies $X \to Y$ in $F^+$ such that $XY \subseteq Z$.

In principle, it is easy to compute $\pi_Z(F)$-just compute $F^+$ and project it onto $Z$. In practice, however, this method is highly intractable, since the number of dependencies in $F^+$ is often exponential in the size of $F$. Also, it is rather the cover that we want to know than the closure of $\pi_Z(F)$ itself. And this issue, even knowing $F^+$, is not trivial. The main purpose of this paper is to construct an expression equivalent to a cover of projected dependencies for any set $F$.

## 2. Functional Dependency as a Boolean Dependency

The notion of Boolean dependency has been used by many authors, e.g. (Berman and Blok, 1985; Demetrovics *et al.*, 1991; Thalheim, 1987b). In this chapter, we shall introduce another interpretation of a functional dependency as a Boolean dependency and use it in the construction of the algorithm.

It is known (Fagin, 1977) that an FD can be viewed as a formula in propositional logic. For instance, if $R = \{A, B, C\}$, then $A \to B$ can be expressed as the first-order sentence

$$(\forall\, a c_1 c_2 b_1 b_2)\Big((R a c_1 b_1 \wedge R a c_2 b_2) \supset b_1 = b_2\Big)$$

where $R$ is the predicate symbol referring to the relevant relation

| | |
|---|---|
| $Rac_1b_1 \wedge Rac_2b_2$ | reads: for any two tuples $A$ does not change |
| $b_1 = b_2$ | reads: $B$ does not change |

Hence the whole formula reads:

for any two tuples, if $A$ does not change, then $B$ does not change

or, equivalently,

for any two tuples, if $B$ changes, then $A$ changes

Let $\varphi : R \to \{0, 1\}$. For any attribute $A \in R$, for any two tuples $t_1, t_2 \in r$,

$$\varphi(A) = \begin{cases} 1 & \text{if} \quad t_1[A] \neq t_1[A] \\ 0 & \text{if} \quad t_1[A] = t_1[A] \end{cases}$$

If $\varphi(A)$ is denoted by $A$, then $A \to B$ can be written as a Boolean formula $B \supset A$ or, equivalently, $A \vee \overline{B}$. We extend $\varphi$ to give the truth assignment to attribute sets by defining, $\varphi(X) = \sum_{A \in X} \varphi(A)$, i.e. $\varphi$ assigns 1 to $X$ iff it assigns 1 to at least one attribute in $X$. A functional dependency $ABC \to DEF$ can thus be written as a Boolean formula $A \vee B \vee C \vee \overline{D}\,\overline{E}\,\overline{F}$. In general, $X \to Y$ (for sets of attributes) will be denoted by $X \vee \overline{Y}$.

Given a set of functional dependencies $F = \{f_1, f_2, \ldots, f_n\}$, a Boolean function $\varphi(R) = f_1 f_2 \cdots f_n$ *represents* $F$ in the following sense: $\varphi(R) = 1$ iff the relation $r$ on $R$ satisfies all dependencies in $F$. If, for some combination of attributes in $R$, $\varphi(R) = 0$, then there exist two tuples in some relation $r$, which violate at least one dependency in $F$. For example, $A \vee \overline{B} = 0$ for $A = 0$ and $B = 1$ means that there exists a pair of tuples in some relation $r$ on $R(A, B, C)$ in which the value of attribute $A$ does not change ($A = 0$), while the value of attribute $B$ changes ($B = 1$). In the sequel, we shall refer to Boolean functions representing $F$ simply by FD's (functional dependencies).

The function $\varphi(R)$ can be regarded as a mapping from the set of $F$ of FD's into the set of Boolean expressions, so we shall also use notation $\varphi(F)$ to denote a Boolean function representing $F$. Every Boolean function $f(A_1, A_2, \ldots, A_n)$ can be expanded along variables $A_1, A_2, \ldots, A_k$, where $1 \leq k \leq n$, in the following way:

$$f(A_1, A_2, \ldots, A_n) = \sum_{i=0}^{2^k - 1} A_1^* A_2^* \cdots A_k^* f_i(A_{k+1}, \ldots, A_n) \tag{1}$$

where each $A_i^*$ is either $A_i$ or $\overline{A_i}$ depending on the current value of index $i$, and each $f_i$ is the value of $f$ for such a combination of values of $A_1, A_2, \ldots, A_k$ $(0, 1)$ that $A_1^* A_2^* \cdots A_k * \equiv 1$. Any form which is a disjunction of terms like in (1) is called a *disjunctive* form.

**Example 1.** $\varphi(A, B, C) = (A \vee \overline{B})(B \vee \overline{C})$ expanded along the variable $B$ becomes $\overline{B}\varphi(B = 0) \vee B\varphi(B = 1) = \overline{B}\,\overline{C} \vee BA$.

**Definition 1.** Let $R = \{A_1, A_2, \ldots, A_n\}$. A *projection* of function $\varphi(R)$ onto the set $X \subseteq R$ denoted by $\pi_X(\varphi)$ is the function $f(X) = \sum_i f_i(X)$, where $f_i$ are the terms of expansion (1) in which $\{A_1, A_2, \ldots, A_k\} = R - X$.

Thus, to compute a projection onto $X$, one must expand a function along the variables in $R - X$. In Example 1, $\pi_{AC}(\varphi) = A \vee \overline{C}$. The name "projection" has been chosen deliberately. A Boolean function can be presented in the *tabular form*, where each column corresponds to one variable, and each row corresponds to one term $f_i$ in (1), when the expansion is taken along all the variables. An entry is equal to 0 or 1 depending on whether the corresponding variable in $A_1^*, A_2^* \cdots A_n^*$ is negated or not. Then $\pi_X(\varphi)$ is a function whose table is obtained from the tabular representation of $\varphi$ by simply deleting columns belonging to $R - X$ and removing duplicate rows. Some properties of $\pi_X(F)$ are given by Lemma 1.

**Lemma 1.**

(a) *For any* $X \subseteq R$, $\varphi \subseteq \pi_X(\varphi)$.

(b) $\pi_X(\varphi(Y)) \equiv \varphi$ *if* $X \cap Y = \emptyset$, *or 1 if* $X = Y$.

(c) $\pi_X(\varphi \vee \gamma) = \pi_X(\varphi) \vee \pi_X(\gamma)$.

(d) $\pi_{XY'}(\varphi(Y,X)\gamma(X,Z)) = \pi_{XY'}(\varphi)\pi_X(\gamma)$, *where* $Y' \subseteq Y$.

(e) *If* $\varphi$ *is written in any disjunctive form, then* $\pi_X(\varphi)$ *is obtained by deleting all variables in* $R - X$ *in the expression of* $\varphi$. *If some term is thereby deleted, then* $\pi_X(\varphi) \equiv 1$.

*Proof.* We shall only prove property (d), because the proof of the others is straightforward.

To compute $\pi_{XY'}(\varphi(Y,X)\gamma(X,Z))$, we must first represent it in the form (1) expanding along variables in $Y''Z$, where $Y'' = Y - Y'$, and then delete these variables. The terms $A_1^*, A_2^* \cdots A_k^*$ in (1) denote now the variables $Y''$ and $Z$ negated or not (e.g. $y_1\overline{y}_2\overline{y}_3\overline{z}_1z_2\overline{z}_3$). Denote these terms by $Y^iZ^j$, where $0 \le i \le 2^{|Y''|} - 1 = N$, and $0 \le j \le 2^{|Z|} - 1 = M$. Then the expansion can be rewritten in the following manner (functions $\varphi_i$ and $\gamma_j$ depend on the variables $XY'$ and $X$, respectively):

$$\varphi(Y,X)\gamma(X,Z) = Y^0\varphi_0\sum_{i=0}^{M} Z^i\gamma_i \vee Y^1\varphi_1\sum_{i=0}^{M} Z^i\gamma_i \vee \cdots \vee Y^N\varphi_N\sum_{i=0}^{M} Z^i\gamma_i$$

and

$$\pi_{XY'}\Big(\varphi(Y,X)\gamma(X,Z)\Big) = \varphi_0\sum_{i=0}^{M}\gamma_i \vee \varphi_1\sum_{i=0}^{M}\gamma_i \vee \cdots \vee \varphi_N\sum_{i=0}^{M}\gamma_i$$

$$= \sum_{j=1}^{N}\varphi_j\sum_{i=1}^{M}\gamma_i = \pi_{XY'(\varphi)\pi_X(\gamma)}$$

■

**Example 2.** The tables for $\varphi(A,B,C) = (A \vee \overline{B})(B \vee \overline{C}) = AB \vee A\overline{C} \vee \overline{B}\,\overline{C}$ and its projection onto $AC$ which is the function $A \vee \overline{C}$ are shown below.

Table $\varphi$

| $A$ | $B$ | $C$ |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 0 | 0 |

Table $\pi_{AC}(\varphi)$

| $A$ | $C$ |
|---|---|
| 1 | 0 |
| 1 | 1 |
| 0 | 0 |

It is interesting to notice that the projection obtained in Example 1 represents an FD $A \to C$ which holds in the projection of $R(A,B,C)$ with the set $F$ of FD's equal to $\{A \to B,\ B \to C\}$. This observation will be formalized in the next section. Also, $A \to C$ does not belong to $F$—it belongs to $F^+$. But $\varphi(F) \subset A \vee \overline{C}$ and this fact is generalized by the following theorem.

**Theorem 1.** *If $\varphi$ represents $F$, then it also represents $F^+$, i.e. $\varphi(F) \equiv \varphi(F^+)$.*

*Proof.* An FD $X \to Y$ not contained in $F$, but contained in $F^+$, can be derived from $F$ by a repeated application of Armstrong's axioms (Armstrong, 1974). Let $\varphi = f_1 f_2 \cdots f_n$ and let $f_1 = Z \vee \overline{V}$. Applying the reflexivity rule to $Z \to V$ we get $f_{n+1} = ZV \to V$, i.e. $Z \vee V \vee \overline{V} = 1$ and hence $\{f_{n+1}\} \cup F$ is represented by $\varphi(F)$, because $\varphi(F) \equiv \varphi f_{n+1}$. In the same way, we prove the theorem for the augmentation rule $(Z \to V \Rightarrow ZW \to VW)$ and transitivity rule $(Z \to V$ and $V \to W \Rightarrow Z \to W)$. Thus any application of Armstrong's axioms yields a new set $F$, but every new set is represented by the same function $\varphi(F)$, which completes the proof. ∎

Theorem 1 has an important consequence: *a function $\varphi(F)$ can be used to derive all members of $F^+$ if we are able to find a method of representing it as a conjunction.*

## 3. Properties of Function $\varphi(F)$

Not every Boolean function represents $F$. One should rather say that most functions do not represent any semantics of a relation schema. For example, $\varphi(A,B,C) = \overline{A}\,\overline{B}\,\overline{C}$ means that in any relation $r$ over $R(A,B,C)$, the values $A,B,C$ never change and thereby are constants—a constraint that is unlikely to be imposed on any relation schema. Similarly, $\varphi(A,B,C) = \overline{A} \vee \overline{B} \vee \overline{C}$ upon closer examination leads to the same conclusion that, in any relation, one (or more) of the attributes may never change. Below we give some primitive constructs which can be used to build valid representations of FD's.

a) Function $\varphi(R) \equiv 1$.
It represents a trivial FD. Indeed, a trivial FD is a statement $R \to R$ which is equivalent to $\varphi(R) = A_1 \vee A_2 \vee \cdots \vee A_n \vee \overline{A}_1\,\overline{A}_2 \cdots \overline{A}_n$ which is 1 irrespective of the values assigned to the variables.

b) $\varphi(R) = A_1 \vee A_2 \vee \cdots A_k$.
It represents a key dependency with $\{A_1, A_2, \ldots, A_k\}$ $(k \geq 1)$ being the key. Indeed, the above expression is a constraint saying that at least one of the attributes of $\{A_1, A_2, \ldots, A_k\}$ must always change, which is a key constraint.

c) $\varphi(R) = A_1 \vee A_2 \vee \cdots A_k \vee \overline{B}_1$.
It represents a dependency $A_1 A_2 \cdots A_k \to B_1$.

d) An expression of the form $\varphi(R) = A_1 A_2 \cdots A_n \vee \overline{A}_1\,\overline{A}_2 \cdots \overline{A}_n$.
It represents the following set of FD's: $\{A_1 \to A_2, \ldots, A_{k-1} \to A_k, A_k \to A_1\}$ The validity of the above can easily be proven (the details are omitted) by induction (for $k = 2$ we have $(A_1 \vee \overline{A}_2)(A_2 \vee \overline{A}_1) = A_1 A_2 \vee \overline{A}_1\,\overline{A}_2$).

**Lemma 2.** *If $\varphi(F)$ represents $F$, then $\overline{\varphi(F)}$ does not represent any FD.*

*Proof.* Since every $\varphi$ can be represented as a conjunction of the formulae a), b) or c), its negation is a disjunction of terms such that each of them contains at least one negated variable. This means that, in any relation $r$, at least one attribute never changes which is not an FD constraint and hence is not in any $F$. ■

From the above lemma we conclude that the function $\varphi(R) \equiv 0$ does not represent any functional dependency since it is a negation of a trivial dependency.

**Lemma 3.** *$\pi_X(\varphi(R))$ always represents some set of FD's (possibly trivial FD's).*

The proof will be given in the next section, where we introduce a more suitable form of representing FD's. Meanwhile, we notice that this lemma formalizes our intuitive association of projections of relations and projections of Boolean functions representing constraints on these relations.

**Theorem 2.** *Let $X_1, X_2, \ldots, X_n$ be subsets of $R$ such that $\bigcup_i X_i = R$. If a Boolean function[1] $\varphi(R)$ can be represented as $\bigcap_i f_i(X_i)$ for some functions $f_i$, then it must also be represented as a conjunction of $\pi_{X_i}(\varphi)$ for all $i$.*

*Proof.* Suppose that $\varphi(R) = \bigcap_i f_i(X_i)$ for some functions $f_i(X_i)$. We will show that this implies $\varphi(R) = \bigcap_i \pi_{X_i}(R)$. It is sufficient to consider decompositions of $\varphi(R)$ into two components, since $\varphi(R) = \bigcap_i f_i(X_i)$ implies $\varphi(R) = f_1(X_1)f_2(X_2)$ for some $X_1$, $X_2$, or alternatively $\varphi(R) = f_1(X,Y)f_2(Y,Z)$ for some $X$, $Y$, $Z$ of which one (and only one) set can be empty. If this is the case, then $\pi_{XY}(\varphi) = \pi_{XY}(f_1 f_2) = \pi_{XY}(f_1)\pi_Y(f_2)$ (property (d) from Lemma 1) and by property (b) from Lemma 1 we get $\pi_{XY}(\varphi) = f_1\pi_Y(f_2)$. Similarly, $\pi_{YZ}(\varphi) = f_2\pi_Y(f_1)$. Hence, from property (a), we get $\pi_{XY}(\varphi)\pi_{YZ}(\varphi) = f_1\pi_Y(f_2)f_2\pi_Y(f_1) = f_1 f_2 = \varphi$. ■

Thus any function $f_i(X_i)$ that might be a part of the conjunction $\bigcap_i f_i(X_i)$ is implied by $\pi_{X_i}(\varphi)$. From Lemma 3, and from the fact that every $\varphi$ is a conjunction of FD's, we may state the following.

**Corollary 1.** *The projection of a Boolean function representing $F$ onto the set $X$ implies all FD's valid in the projection of $F$ onto $X$, and thus it is the cover for the projected dependencies.*

This statement is a well-known fact (e.g. Ullman, 1989) when talking about the closure of $F$ projected onto $X$. However, we avoid intractability of dealing with closures by manipulating $F$ only, represented as a Boolean expression.

[1] Not necessarily representing an FD.

**Example 3.** We will find the projected dependencies holding in the projection of $R(A,B,C,D)$ onto $\{A,B\}$ for $F = \{A \to B,\ B \to C,\ C \to D,\ D \to A\}$.

$$\varphi(F) = (A \vee \overline{B})(B \vee \overline{C})(C \vee \overline{D})(D \vee \overline{A}) = ABCD \vee \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

$$\pi_{AB}(\varphi) = (\text{deleting } C, D)\ AB \vee \overline{A}\,\overline{B} = (A \vee \overline{B})(B \vee \overline{A})$$

Thus $\{A \to B,\ B \to A\}$ is the cover for FD's holding in $\pi_{AB}(F)$. The dependency $B \to A$ is redundant in $F$ but is not redundant in $\pi_{AB}(F)$. This illustrates the inclusion $\pi_{AB}(F) \supseteq$ {the set of functions of variables $A$ and $B$ that can appear in the conjunctive decomposition of $\varphi(F)$}.

This method, although simple, is unattractive from the computational point of view. First, we must transform the function $\varphi(F)$ to the disjunctive form, from which we immediately get $\pi_X(\varphi)$, but then we must transform it back to the conjunctive form to pick up dependencies we are interested in. Both steps are highly intractable and hence the question is raised if there exists an algorithmic way of overcoming at least some of the difficulties. The solution is presented in the next section.

## 4. Algorithm to Compute $\pi_X(F)$

At first we notice that rather than using $\varphi(F)$ we may just use the function $\overline{\varphi(F)}$ which is uniquely determined as $\overline{\varphi(F)} = \overline{f}_1 \vee \overline{f}_2 \vee \cdots \vee \overline{f}_n$, where each $\overline{f}_i$ is now of the form $\overline{A}_1\,\overline{A}_2 \cdots \overline{A}_k B$, in which all but one variables are negated. For $\varphi$ from Example 3, $\overline{\varphi(F)} = \overline{A}B \vee \overline{B}C \vee \overline{C}D \vee \overline{D}A$. This form is suitable for computing projections. However, it is not true that $\pi_X(\varphi) = \overline{\pi_X(\overline{\varphi})}$. Thus we cannot just project $\overline{\varphi(F)}$ and take its negation to obtain $\pi_X(\varphi)$. Indeed, $\pi_X(\varphi) = \overline{\overline{\varphi_0 \vee \varphi_1 \vee \cdots \vee \varphi_N}}$, where each $\varphi_i$ is from expansion (1) of $\varphi$ along the variables $R - X$, and hence

$$\pi_X(\varphi) = \overline{\overline{\varphi}_0\,\overline{\varphi}_1 \cdots \overline{\varphi}_N} \tag{2}$$

Therefore, to obtain the projection of $\varphi$ knowing $\overline{\varphi}$, we would need to compute $N+1$ functions $\overline{\varphi}_i$, where $N = 2^{|R-X|} - 1$, which is of course intractable. We will show that formula (2) can be computed in $|R - X|$ steps only.

**Lemma 4.** *Denote $\overline{\varphi}_0, \overline{\varphi}_1, \ldots, \overline{\varphi}_N$ by $f_0, f_1, \ldots, f_N$, respectively. The recursive version of formula (2) is $\pi_X(\varphi) = \overline{f_0^k f_1^k}$, where*

a) $f_0^k = f_0^{k-1} f_1^{k-1}|_{A_k=0}$ *and* $f_1^k = f_0^{k-1} f_1^{k-1}|_{A_k=1}$, (3)

b) *$k$ is the number of attributes in $R - X = \{A_1, \ldots, A_k\}$,*

c) *$f_0^0 f_1^0$ is the function $\varphi$ (expansion along a variable $A_0$ on which the function does not depend).*

*Proof.* If $R - X$ contains only one variable ($k = 1$), then $\pi_X(\varphi) = \varphi_0^1 \vee \varphi_1^1 = \overline{f_0^1 f_1^1}$. Suppose that, for some $k$, $\pi_{X-A_{k+1}}(\varphi) = \overline{f_0^k f_1^k}$, where $f_0^k = f_0^{k-1} f_1^{k-1}|_{A_k=0}$ and $f_1^k = f_0^{k-1} f_1^{k-1}|_{A_k=1}$. Then expanding $\overline{f_0^k f_1^k}$ along $A_{k+1}$ we get

$$\overline{f_0^k f_1^k} = \overline{A}_{k+1}\overline{f_0^k(A_{k+1} = 0) f_1^k(A_{k+1} = 0)} \vee A_{k+1}\overline{f_0^k(A_{k+1} = 1) f_1^k(A_{k+1} = 1)}$$

and

$$\pi_X(\varphi) = \overline{f_0^k(A_{k+1}=0) f_1^k(A_{k+1}=0)} \vee \overline{f_0^k(A_{k+1}=1) f_1^k(A_{k+1}=1)}$$

$$= \overline{f_0^k(A_{k+1}=0) f_1^k(A_{k+1}=0) f_0^k(A_{k+1}=1) f_1^k(A_{k+1}=1)} = \overline{f_0^{k+1} f_1^{k+1}}$$

which completes the proof by induction. ■

In Example 3,

$$X = \{A, B\},\ A_1 = C,\ A_2 = D,\ f^0 = \overline{A}B \vee \overline{B}C \vee \overline{C}D \vee \overline{D}A,\ f_0^1 = f(C=0),$$

$$f_1^1 = f(C=1),\ f_0^2 = f(C=0)f(C=1)|_{D=0},\ f_1^2 = f(C=0)f(C=1)|_{D=1}$$

Using formula (3), we must perform $k$ iterations, where $k$ is the number of attributes in $R - X$ to compute a projection. The function $\overline{\varphi(F)}$ can be represented by a table with columns $A_1, \ldots, A_k$ and only $r$ rows, $r$ being the number of terms in the disjunctive form, where each entry is either 0, 1, or $x$ depending on whether the corresponding variable is negated, not negated, or does not appear, respectively. Such a table represents dependencies given in the *canonical form* (Ullman, 1989), i.e. with only single attributes on the right-hand side (e.g. $A \to BCD$ is rewritten as a set $\{A \to B,\ A \to C,\ A \to D\}$ corresponding to $\overline{A}B \vee \overline{A}C \vee \overline{A}D$). For example, $\varphi(F) = \overline{A}_1A_2 \vee \overline{A}_2A_3 \vee \overline{A}_3A_4 \vee \overline{A}_4A_1$ is represented by

| $A_1$ | $A_2$ | $A_3$ | $A_4$ |
|---|---|---|---|
| 0 | 1 | $x$ | $x$ |
| $x$ | 0 | 1 | $x$ |
| $x$ | $x$ | 0 | 1 |
| 1 | $x$ | $x$ | 0 |

(4)

From now on, we shall omit the symbol of negation in the function $\overline{\varphi}$ considering it as a function representing $F$. Functions with subscripts 0 and 1 appearing in (3) are selections made from the table, such that the value of the attribute is either 0 or 1. For example, $\varphi_0^1$ is the first row of the above table (the variable $A_1$ is 0); $\varphi_1^1$ is the last row (the variable $A_1$ is 1). An informal description of projection onto $X$ is as follows:

1. Select the first attribute in $R - X$ (*current attribute*).
2. Split the function table (*current table*) into three tables $f_0, f_1, f_x$ containing the rows from the original table which have respectively 0,1, or $x$ in the column for the current attribute. These tables do not contain a column corresponding to the current attribute.

3. 'Multiply' (row by row) tables $f_0, f_1$ in accordance with the rules given in table (5) (if one of these tables is empty, then the result is null; if one of the tables contains a row of '$x$', then the result is the other table[2]), where $\Lambda$ means a null value causing the result of the 'product' to be null. It is a Boolean conjunction of a variable and its negation.
4. Append the resultant table to table $f_x$ and make it the current table. Make the next attribute in $R - X$ the current attribute and repeat steps 1–4 until all attributes in $R - X$ have been exhausted. The algorithm gives only trivial dependencies if at some stage we get the table $f_0$ or $f_1$ only.

| * | 0 | 1 | $x$ |
|---|---|---|---|
| 0 | 0 | $\Lambda$ | 0 |
| 1 | $\Lambda$ | 1 | 1 |
| $x$ | 0 | 1 | $x$ |

(5)

The last *current table* contains only columns for $X$ and represents $\pi_X(F)$. If we get a null current table at some stage, then the process is aborted, as the null table corresponds to the function 0 representing trivial dependencies. Having introduced another representation of functional dependencies and a method of obtaining the projection we can now prove Lemma 3.

*Proof of Lemma 3.* It is sufficient to prove that by the above method we shall never get a row that does not represent a functional dependency. Such a row should consist of more than one '1' or contain no 0's. Since $f_0$ contains no 1's and $f_1$ has only one '1', the first possibility cannot occur. But a zero cannot be eliminated according to (5), unless we eliminate an entire row. ■

**Example 4.** (only trivial projected dependencies)
Let us take $F = \{A_1 \to A_2,\ A_3A_4 \to A_1,\ A_4 \to A_2\}$. Clearly, no dependencies hold in $X = \{A_2, A_3\}$. Attributes in $R - X$ are $A_1$ and $A_4$. The first run of the loop will give the tables:

$f_0^1$

| $A_2$ | $A_3$ | $A_4$ |
|---|---|---|
| 1 | $x$ | $x$ |

$f_1^1$

| | | |
|---|---|---|
| $x$ | 0 | 0 |

$f_x^1$

| | | |
|---|---|---|
| 1 | $x$ | 0 |

[2] Such a row induces the function 1.

The new current function becomes

| $A_2$ | $A_3$ | $A_4$ |
|---|---|---|
| 1 | 0 | 0 |
| 1 | $x$ | 0 |

and the next run will produce only table $f_0^4$ indicating no dependencies in $\pi_{A_2A_3}(F)$.

**Example 5.** Find $\pi_{A_2A_3A_6}(F)$ for the following set:

$$F = \left\{ \begin{array}{l} A_1A_2 \to A_3,\ A_3 \to A_1,\ A_2A_3 \to A_4,\ A_4 \to A_5,\ A_4 \to A_6, \\ A_2A_5 \to A_1,\ A_2A_5 \to A_3,\ A_3A_6 \to A_2,\ A_3A_5 \to A_6 \end{array} \right\}$$

$f_0^1$

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | $x$ | $x$ | $x$ |

$f_1^1$

| | | | | | |
|---|---|---|---|---|---|
| 1 | $x$ | 0 | $x$ | $x$ | $x$ |
| 1 | 0 | $x$ | $x$ | 0 | $x$ |

$f_x^1$

| | | | | | |
|---|---|---|---|---|---|
| $x$ | 0 | 0 | 1 | $x$ | $x$ |
| $x$ | $x$ | $x$ | 0 | 1 | $x$ |
| $x$ | $x$ | $x$ | 0 | $x$ | 1 |
| $x$ | 0 | 1 | $x$ | 0 | $x$ |
| $x$ | 1 | 0 | $x$ | $x$ | 0 |
| $x$ | $x$ | 0 | $x$ | 0 | 1 |

$f_0^1 f_1^1 \vee f_x^1$

| $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ |
|---|---|---|---|---|
| 0 | 0 | 1 | $x$ | $x$ |
| $x$ | $x$ | 0 | 1 | $x$ |
| $x$ | $x$ | 0 | $x$ | 1 |
| 0 | 1 | $x$ | 0 | $x$ |
| 1 | 0 | $x$ | $x$ | 0 |
| $x$ | 0 | $x$ | 0 | 1 |

split: $f_0^4$

| $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ |
|---|---|---|---|---|
| $x$ | $x$ | 0 | 1 | $x$ |
| $x$ | $x$ | 0 | $x$ | 1 |

$f_1^4$

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 1 | $x$ | $x$ |

$f_x^4$

| | | | | |
|---|---|---|---|---|
| 0 | 1 | $x$ | 0 | $x$ |
| 1 | 0 | $x$ | $x$ | 0 |
| $x$ | 0 | $x$ | 0 | 1 |

$f_0^4 f_1^4 \vee f_x^4$

| $A_2$ | $A_3$ | $A_5$ | $A_6$ |
|---|---|---|---|
| 0 | 0 | 1 | $x$ |
| 0 | 0 | $x$ | 1 |
| 0 | 1 | 0 | $x$ |
| 1 | 0 | $x$ | 0 |
| $x$ | 0 | 0 | 1 |

split: $f_0^5$

| $A_2$ | $A_3$ | $A_5$ | $A_6$ |
|---|---|---|---|
| 0 | 1 | 0 | $x$ |
| $x$ | 0 | 0 | 1 |

$f_1^5$

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | $x$ |

$f_x^5$

| | | | |
|---|---|---|---|
| 0 | 0 | $x$ | 1 |
| 1 | 0 | $x$ | 0 |

$f_0^5 f_1^5 \vee f_x^5$

| $A_2$ | $A_3$ | $A_6$ | |
|---|---|---|---|
| 0 | 0 | 1 | |
| 0 | 0 | 1 | duplicate the row |
| 1 | 0 | 0 | |

The result is $\overline{A}_2\overline{A}_3A_6 \vee \overline{A}_3\overline{A}_6A_2$ representing $A_2A_3 \to A_6$ and $A_3A_6 \to A_2$. The dependency $A_3A_6 \to A_2$ is in $F$, whereas $A_2A_3 \to A_6$ can be inferred from $A_2A_3 \to A_4$ and $A_4 \to A_6$.

This method, although conceptually simple, is not guaranteed to run in polynomial time. We will now rectify it by placing one term $\overline{X}Y$ of $\varphi$ in one row of the table. Thereby, some rows will represent disjunction of several rows in the original table. We call this representation an *NC-form* (non-canonical form) or *NC-table*. A formal definition of an NC-row is now introduced.

**Definition 2.** Two rows $r_1$ and $r_2$ are equivalent to one *NC-row* iff they agree on all '0' values. In such a case, an NC-row is constructed by replacing all $x$'s in one row by the corresponding values of the other row. Such an operation is called *recombination.*

The opposite operation (splitting one NC-row into a number of canonical rows) is called *decomposition.* We will also need *partial decomposition*, i.e. breaking down one NC-row into a number of rows of which some may be NC-rows. For example, $(1, 1, 1, 0, 0, x, x)$ is equivalent to three canonical rows: $(1, x, x, x, 0, 0, x, x)$, $(x, 1, x, 0, 0, x, x)$, $(x, x, 1, 0, 0, x, x)$ (decomposition), but also to the pair: $(1, 1, x, 0, 0, x, x)$, $(x, x, 1, 0, 0, x, x)$ (partial decomposition).

In Example 5, the table is as follows:

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ | |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | $x$ | $x$ | $x$ | |
| 1 | $x$ | 0 | $x$ | $x$ | $x$ | |
| $x$ | 0 | 0 | 1 | $x$ | $x$ | |
| $x$ | $x$ | $x$ | 0 | 1 | 1 | two NC-rows equivalent |
| 1 | 0 | 1 | $x$ | 0 | $x$ | to four canonical rows |
| $x$ | 1 | 0 | $x$ | $x$ | 0 | |
| $x$ | $x$ | 0 | $x$ | 0 | 1 | |

For NC-tables, new rules of multiplication must be derived, as well as a new structure of the table must be introduced. The algorithm of stepwise projection tends to produce strongly redundant dependencies, thereby leading to exponential complexity.

**Example 6.** Consider the set $A_i \to X$, $X \to Y$ for $1 \leq i \leq m$. Obviously,

$$\pi_{AY}(F) = \left\{A_i \to Y,\ 1 \leq i \leq m\right\}\left(A = \bigcup_1^m A_i\right)$$

By rewriting these dependencies in the canonical form we shall obtain $m|X| + |Y|$ dependencies. If we apply successively steps 1–4 of the original algorithm, the final result will be strongly redundant with $2^m - 1$ dependencies $L \to Y$ for every subset $L$ of $\{A_i\}$. The NC-table shown below gives some hints regarding the direction in which the modification of the algorithm should go: the table must be structured in a way which provides more control over the multiplication of rows. Relying only on one criterion of selecting the current attribute, and then processing rows at random

$\longleftarrow R - X \longrightarrow$

| $A_1$ | $A_2$ | ... | $A_m$ | $X_1$ | $X_2$ | ... | $X_k$ | $Y_1$ | $Y_2$ | ... | $Y_n$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $x$ | ... | $x$ | 1 | 1 | ... | 1 | $x$ | $x$ | ... | $x$ |
| $x$ | 0 | ... | $x$ | 1 | 1 | ... | 1 | $x$ | $x$ | ... | $x$ |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| $x$ | $x$ | ... | 0 | 1 | 1 | ... | 1 | $x$ | $x$ | ... | $x$ |
| $x$ | $x$ | ... | $x$ | 0 | 0 | ... | 0 | 1 | 1 | ... | 1 |

surely leads to redundancies, and in consequence—to exponential complexity. Let us introduce the following observations:

a) A table may have rows with only '$x$' symbols in the columns belonging to $R - X$. It represents projected dependencies which are already in $F$. All rows satisfying this condition are written in *RESULT*.

In Example 5 a row of *RESULT* ($A_3A_6 \rightarrow A_2$) is shown below:

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ |
|---|---|---|---|---|---|
| $x$ | 1 | 0 | $x$ | $x$ | 0 |

b) If there are no rows that can be written in *RESULT*, then the table which produces non-trivial projected dependencies must have at least one row with only '0' and '$x$' symbols in the columns belonging to $R - X$ and at least one '1' in a column belonging to $X$.

Indeed if there is no such row, the function $\overline{\varphi}_0$ in (2) is 0, and the whole expression yields 1. Let $f_0 = \{r_0\}$ denote a set of such rows. A formal definition is as follows:

- $f_0$ is a set of rows that have at least one '0' in the columns $R - X$ and has no 1's in any column belonging to $R - X$. Rows having 1's in the columns $R - X$ and $X$ are subjected to partial decomposition: all 1's in the columns are replaced by '$x$' and such a row goes to $f_0$, and all 1's in the columns $X$ are replaced by '$x$' and such a row goes to a new table $f_{0,1}$. Hence the table $f_0$ may contain all symbols in the columns belonging to $X$.

Table $f_0$ from Example 5 is shown below:

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | $x$ | $x$ | $x$ |
| $x$ | $x$ | $x$ | 0 | $x$ | 1 |
| $x$ | $x$ | 0 | $x$ | 0 | 1 |

The algorithm must remove all zeroes from at least one row of $f_0$ to produce non-trivial dependencies.

c) Every table which produces non-trivial projected dependencies must have at least one row with only '1' and '$x$' symbols in the columns belonging to $R - X$ and at least one '0' in a column belonging to $X$. Indeed, if there is no such row, the function $\overline{\varphi}_N$ in (2) is 0, and the whole expression yields 1. Let $f_1 = \{r_1\}$ denote a set of such rows. A formal definition is as follows:

- $f_1$ is a set of rows that contain at least one '1' in a column belonging to $R-X$ and has no '0' in any column belonging to $R-X$. Rows having 1's in columns belonging to $X$ are subjected to partial decomposition so that all 1's in the columns belonging to $X$ are replaced by $x$'s and such rows are written in $f_1$, while all 1's in the columns belonging to $R-X$ are replaced by $x$'s, and such a row is written in the *RESULT*. Thus $f_1$ contains only 0's and $x$'s in the columns belonging to $X$.

Table $f_1$ from Example 5 is shown below:

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ |
|---|---|---|---|---|---|
| 1 | $x$ | 0 | $x$ | $x$ | $x$ |
| $x$ | 0 | 0 | 1 | $x$ | $x$ |

d) Every dependency $X_i \to Y_i \in \pi_X(\varphi)$ must satisfy the condition

$$\exists(r_1 \in f_1),\, r_1(A) = 0 \Rightarrow A \in X_i \quad \text{and} \quad \exists(r_0 \in f_0)\forall(B \in Y_i),\, r_0(B) = 1$$

To prove this assertion, we notice that $\overline{\varphi}_0\overline{\varphi}_N$ in (2) is equivalent to $\pi_X(f_0) \bullet \pi_X(f_1)$, where $\pi_X$ are projections (in the relational sense) of tables $f_0$ and $f_1$ onto $X$, and $\bullet$ is a modified operation (5) applied to NC-rows. Furthermore, $\overline{\varphi}_0\overline{\varphi}_N$, implies the whole expression (2) and is an FD at all times, whereas no other sub-expression (also implied by (2)) is guaranteed to be an FD. In other words, the left-hand side of any dependency in $\pi_X(\varphi)$ must contain all attributes corresponding to a 0-column of some row of $f_1$, all the right-hand side attributes must be found in some row of $f_0$ (which is obvious as $f_0$ is the only table containing 1's in columns $X$). Hence the algorithm should always try to 'remove' zeroes of $r_0$ by ones of $r_1$ before 'removing' zeroes of $f_{0,1}$ (ref. (b)) in an attempt to produce more ones (i.e. producing more rows $r_1$). Zeros are "removed" by multiplication of rows. With canonical rows, we can 'remove' only one zero (for the current attribute) at a time. With NC-rows, we can 'remove' multiple zeroes in one multiplication. Moreover, we never 'produce' new zeroes in the columns $R - X$, which may happen in the canonical form. This is because of the observation that the 'removal' of zeroes can be accomplished only by rows in $f_1$. In Example 6, it seems obvious that every row of $f_1$ is capable of 'removing' all zeroes from the row of $f_0$ thereby producing an outright result. The rule $1 \bullet 0 = x$ for every pair of attributes in $R - X$ is sound, as it results from the implication: $X \to Y,\ YZ \to V \Rightarrow XZ \to V$. When multiplying $f_1 \bullet f_0$ we may face a pair $(0,1)$ in a column belonging to $X$. Let $0 \bullet 1 = 0$. Denote $r_1^0 = r_1 \bullet r_0$ and introduce an extra rule: $r_1^0 = \Lambda$ iff $\forall(A \in X),\ r_1^0(A) \neq 1$. In other words, the result is null if there is no '1' in any column belonging to $X$. These two rules are sound because of the following implications: $XA \to Y,\ YZ \to AV \Rightarrow XAZ \to V$ and if $V = \emptyset$, then the conjunction of $r_1$ and $r_0$ produces a trivial dependency. Other rules for multiplying $r_1 \bullet r_0$ are summarized in table (6). The soundness thereof can be proven in a similar manner.

$r_0$

| • | 0 | 1 | $x$ |
|---|---|---|---|
| 0 | 0 | $x$ | 0 |
| 1 | $x$ | | $x$ |
| $x$ | 0 | 1 | $x$ |

(6)

e) A table *may have* other rows $r_{0,1}$ (ref. (b)) which satisfy the following definition:

- $f_{0,1}$ is a set of rows that have at least one '1' and at least one '0' in the columns $R-X$, and has no 1's in the columns $X$. Rows having 1's in the columns $R-X$ and $X$ are partially decomposed into tables $f_0$ and $f_{0,1}$. Hence table $f_{0,1}$ may contain only 0's and $x$'s in the columns of $X$

There is no table $f_{0,1}$ in Example 7, whereas in Example 6, it looks as follows:

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ |
|---|---|---|---|---|---|
| $x$ | $x$ | $x$ | 0 | 1 | $x$ |
| 1 | 0 | $x$ | $x$ | 0 | $x$ |

For every current attribute $A$ of $R - X$, all rows of $f_0$ (such that $r_0(A) = 0$) are multiplied by *all rows* of $f_1$ (such that $r_1(A) = 1$) before multiplication $f_1 \bullet f_{0,1}$ for the same current attribute commences, which is performed according to the following table:

$r_{0,1}$

| • | 0 | 1 | $x$ |
|---|---|---|---|
| 0 | 0 | | 0 |
| 1 | $x$ | $x$ | $x$ |
| $x$ | 0 | 1 | $x$ |

(7)

with an extra rule: $r_1^{0,1} = \Lambda$ iff $\forall(A \in R - X),\ r_1^{0,1}(A) = x$, i.e. the rows must have at least one '1' in the columns $R - X$. The rows of $f_1 \bullet f_{0,1}$ which have no 0's in the columns $R - X$ will be added to the rows of $f_1$ for the next current attribute. Notice that 1's covered by 1's of $r_1$ are not counted (they are replaced by '$x$'). This mechanism coupled with the strategy 'first $f_1 \bullet f_0$, then $f_1 \bullet f_{0,1}$' will contribute to the reduction of possible redundancies from the one hand, and the number of rows in $f_1$, from the other. The rule $1 \bullet 1 = 1 \bullet x = x$ says that 1's in $r_1$ can only contribute to the creation of new 1's which are not in $r_1$, because 1's in $r_1$ have already been utilized to eliminate 0's in $r_0$. By definition, there is no combination (0,1) when multiplying rows of $f_1$ and $f_{0,1}$. The soundness of other rules (7) can be proven in a way similar to rules (6). Rows of $f_1$ will be used throughout the algorithm upon removing the current attribute (i.e. setting its value to '$x$').

## 5. Implementation Issues

Tables $f_0, f_1, f_{0,1}$ and $RESULT$ are represented as *two-way lists* of records whose first element is a one-dimensional array. The other elements of the record will be described. The attributes are mapped onto the set $ATTR = \{1, \ldots, n\}$ with first $k$ integers representing attributes of $R - X$. Let $A_0, A_{0,1}$, and $A_1$ denote sets of attributes $A$ in $R - X$ such that we have respectively

$$\exists\,(r_0), r_0[A] = 0,\ \exists\,(r_{0,1}), r_{0,1}[A] = 0 \text{ and } \exists\,(r_1), r_1[A] = 1$$

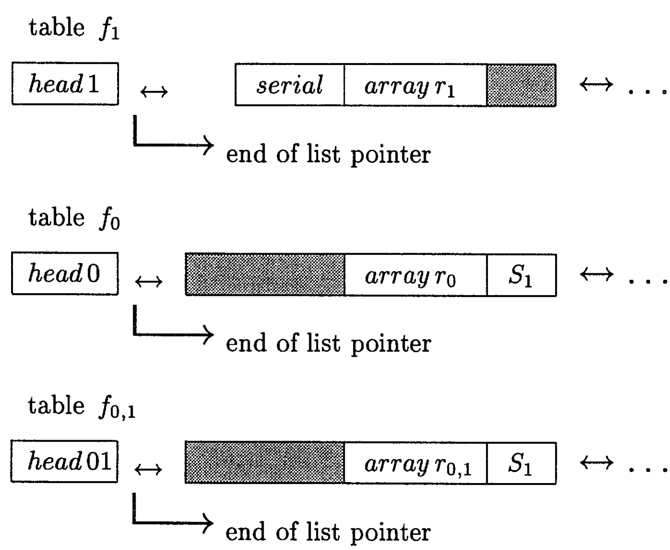
Define two arrays $AR_0$ and $AR_{0,1}$ such that $AR_0[i](AR_{0,1}[i]) = m$ if $i \in R - X$ is equal to 0 in $m$ rows $r_0(r_{0,1})$. Similarly, the array $AR_1$ will store the information how many rows $r_1$ satisfy $r_1[i] = 1$. Such arrays and sets can be easily created/updated when processing the rows. In Example 5, these sets are $A_0 = \{1, 4, 5\}$, $A_{0,1} = \{4, 5\}$, $A_1 = \{1, 4\}$, and the arrays are $AR_1[1] = AR_1[4] = 1$, $AR_0[1] = AR_0[4] = AR_0[5] = 1$, $AR_{0,1}[4] = AR_{0,1}[5] = 1$.

The algorithm must have some strategy of selecting the current attribute $A$: if, at some stage, $A_1 \cap A_0 \neq \emptyset$, then

- $A$ is an attribute in $A_1 \cap A_0 - A_{0,1}$ for which the value of $AR_1[A] \times AR_0[A]$ is minimum;
- if the difference is an empty set, then $A$ is an attribute in $A_1 \cap A_0 \cap A_{0,1}$ for which the value of $AR_1[A] \times AR_{0,1}[A]$ is minimum.

In the presence of several attributes satisfying the minimum criterion, any of them is selected.

Both criteria allow one to defer multiplication of $f_1 \bullet f_{0,1}$ until it is absolutely necessary (i.e. when $A_1 \cap A_0 = \emptyset$, in which case the criterion $AR_1[A] \times AR_{0,1}[A] = \min$ is used, or $A_1 \cap A_0 \cap A_{0,1} \neq \emptyset$, in which case the criterion $AR_1[A] \times AR_{0,1}[A] = \min$ is used). In Example 5, *current* $= \{1\}$. If $A_1 \cap A_{0,1} = \emptyset$, then the algorithm terminates (if $A_1 \cap A_0 = \emptyset$) or continues without rows of $f_{0,1}$ (if $A_1 \cap A_0 \neq \emptyset$). Each row of $f_1$ will be identified by a *serial number* drawn from the pool of integers. Initially, they are assigned successive numbers, starting with 1. The quantity $\max(f_1)$ will be stored so that a new row of $f_1$ (as a result of some product $r_1 \bullet r_{0,1}$) will be assigned the first available number. The numbering system is a part of the strategy to avoid redundancies and unnecessary multiplications. Suppose that at some stage $r_1' \bullet r_0 \in RESULT$ and $r_1'' \bullet r_0 = r_1^0 \notin RESULT$. The rows $r_1'$ and $r_1^0$ may be processed in some later step, and will definitely produce a redundant result because if $r_1' \bullet r_0 \in RESULT$, then $r_1' \bullet (r_1'' \bullet r_0 \in RESULT$. Assign to each row $r_0$ the set $S_1$ containing serial numbers of all rows $r_1$ that produced a result with $r_0$. Two rows $r_1, r_0$ at some stage are subject to multiplication only if $serial(r_1) \notin S_1(r_0)$. The set $S$ will constitute the next element of the record. The same mechanism will be used for multiplication $f_1 \bullet f_{0,1}$. Here, $r_1 \bullet r_{0,1}$ can either produce a new row $r_1$ (analogous to *result*), or a new row $r_{0,1}$ (analogous to a new row $r_0$). Every row $r_{0,1}$ will be assigned the set $S_1$ containing serial numbers of all rows $r_1$, that produced a new row $r_1$ with $r_{0,1}$. Two rows $r_1, r_{0,1}$ at some stage are subject to multiplication only if $serial(r_1) \notin S_1(r_{0,1})$. Thus the lists will have the following structure:




Initially, the above structure represents the set $F$ of functional dependencies to be projected. While they are created, sets $A_1, A_0, A_{0,1}$ and arrays $AR$ are built, serial numbers generated, and a current attribute $A$ selected. The attribute $A$ is then removed from each set $A$ and its counter in the array $AR$ is set to 0. Next, the lists are traversed: rows $\{r_1 | r_1[A] = 1\}$ are attached to the current list $f_1^c$ and deleted from the list $f_1$; in the same manner, the rows $r_0$ and $r_{0,1}$ are processed forming current lists $f_0^c$ and $f_{0,1}^c$, respectively[1].

Each row of $f_0^c$ is multiplied by all rows of $f_1^c$ before the next row of $f_0^c$ is processed. In our example, $(\underline{1}, x, 0, x, x, x) \bullet (\underline{0}, 0, 1, x, x, x)$ gives a null result. A product of two rows updates the corresponding array $AR$ by decreasing the counters

[1] The current lists are introduced to clarify the algorithm. Actually, they are lists of pointers to elements of $f_0, f_1, f_{0,1}$.

of attributes that are replaced by '$x$'. Should the counter reach 0, the attribute is deleted from the set $A$. Only after $f_1^c \bullet f_0^c$ has been completed, rows of $f_{0,1}^c$ are subject to multiplication (not the case in Example 5). However, it may happen that, at this point, the list $f_0$ is empty (all rows of $f_0$ were copied to $f_0^c$, but no product of $f_1^c \bullet f_0^c$ has been appended). This indicates the end of algorithm, because new rows of $f_1$ which would possibly be created from rows $f_1^c$ and $f_{0,1}^c$ cannot be used any more. The rows of $f_1^c$ which, except for the current attribute, have no more 1's, are deleted from the list $f_1^c$ (row $(1, x, 0, x, x, x)$). All other rows of $f_1^c$ are attached to the list $f_1$ at the end of an iteration for the current attribute. Non-null products $r_1 \bullet r_0$ are appended to the end of lists $RESULT$ or $f_0$ (the set $S_1$ of $r_0$ is copied to the new row $r_1 \bullet r_0$ and $r_0$ is deleted); non-null products $r_1 \bullet r_{0,1}$ are appended to the end of lists $f_1$ (their sets $S_1$ are nullified and the rows receive serial numbers) or $f_{0,1}$ (the set $S_1$ of $r_{0,1}$ is copied to the new row $r_1 \bullet r_{0,1}$ and $r_{0,1}$ is deleted). The list $f_{0,1}$ is deleted if at some stage $A_1 \cap A_{0,1} = \emptyset$.

When an iteration for the current attribute is completed, the memory occupied by the current lists (more precisely, by the pointers to these lists) is released, the new current attribute $A$ (in our example $A_4$) selected and an iteration for the new attribute of $R - X$ commences. The product $(x, 0, 0, \underline{1}, x, x) \bullet (x, x, x, \underline{0}, x, 1)$ will produce $(x, x, x, 0, 0, 1) \in RESULT$, the product of $(x, 0, 0, \underline{1}, x, x) \bullet (x, x, x, \underline{0}, 1, x) \in f_{0,1}$ will produce a new row $r_1 = (x, x, x, x, 1, x)$ updating the set $A_1$. The third run for the attribute $A_5$ will produce $(x, x, x, 0, 0, 1)$ already in $RESULT$.

As already mentioned, the algorithm terminates if one of the following conditions is met: (a) $A_1 \cap A_0 = \emptyset$ and $A_1 \cap A_{0,1} = \emptyset$, or (b) the list $f_1^c$ or $f_0^c$ is empty. This condition is always checked after $f_1^c \bullet f_0^c$ is completed and before $f_1^c \bullet f_{0,1}^c$ commences.

If the list $f_1^c$ is empty, then definitely condition (a) is met. However, we may have an empty list $f_0^c$ and $A_1 \cap A_{0,1} \neq \emptyset$ which would lead to unnecessary operations generating dependencies which cannot belong to $\pi_X(F)$.

Example 6 seems trivial for NC-tables: it gives a non-redundant cover after the first iteration. Example 5 turned out very simple, too. Four multiplications (instead of six in the canonical form) were enough to produce the result.

**Example 7.** $R = \{A_1, \ldots, A_{10}\}$, $X = \{A_7, \ldots, A_{10}\}$, $R - X = \{A_1, \ldots, A_6\}$

$$F = \{A_7 \to A_1A_2,\ A_8 \to A_1A_2A_3A_6,\ A_9 \to A_1A_3A_4,\ A_1A_2A_7 \to A_3A_4,\ A_1A_3A_8 \to A_4,$$
$$A_1A_4A_{10} \to A_5A_6,\ A_1A_2A_3A_5A_6 \to A_9A_{10},\ A_1A_2A_4 \to A_8A_9,\ A_1A_3A_6 \to A_7\}$$

This example will illustrate some optimization techniques aiming at reducing the number of multiplications necessary to generate a result (we omit sets $S_1$ in $f_0$ and $f_{0,1}$ as they remain empty). We have

$$f_1 = \left\{ \begin{array}{l} [1], 1, 1, x, x, x, x, 0, x, x, x \\ [2], 1, 1, 1, x, x, 1, x, 0, x, x \\ [3], 1, x, 1, 1, x, x, x, x, 0, x \end{array} \right\}, \quad f_0 = \left\{ \begin{array}{l} 0, 0, 0, x, 0, 0, x, x, 1, 1 \\ 0, 0, x, 0, x, x, x, 1, 1, x \\ 0, x, 0, x, x, 0, 1, x, x, x \end{array} \right\}$$

$$f_{0,1} = \left\{ \begin{array}{l} 0,0,1,1,x,x,0,x,x,x \\ 0,x,0,1,x,x,x,0,x,x \\ 0,x,x,0,1,1,x,x,x,0 \end{array} \right\}$$

$$A_0 = \{1,2,3,4,5,6\},\ \ A_1 = \{1,2,3,4,6\},\ \ A_{0,1} = \{1,2,3,4\}$$

$$AR_0 = \{3,2,2,1,1,2\},\ \ AR_1 = \{3,2,2,1,0,1\},\ \ AR_{0,1} = \{3,1,1,1,0,0\}$$

$$A_1 \cap A_0 = \{1,2,3,4,6\},\ \ A_1 \cap A_0 - A_{0,1} = \{6\}$$

so the attribute $A_6$ is selected for the first iteration. Consequently,

$$f_1^c = \left\{ [2],1,1,1,x,x,1,x,0,x,x \right\},\quad f_0^c = \left\{ \begin{array}{l} 0,0,0,x,0,0,x,x,1,1 \\ 0,x,0,x,x,0,1,x,x,x \end{array} \right\}$$

$$f_1^c \bullet f_0^c = \left\{ \begin{array}{l} x,x,x,x,0,x,x,0,1,1 \\ x,x,x,x,x,x,1,0,x,x \end{array} \right\}$$

The first result is $A_8 \to A_7$ produced by row no. 2 with the second row of $f_0^c$. The new lists are (positions $A_6$ replaced by '$x$')

$$f_1 = \left\{ \begin{array}{l} [1],1,1,x,x,x,x,0,x,x,x \\ [2],1,1,1,x,x,x,x,0,x,x \\ [3],1,x,1,1,x,x,x,x,0,x \end{array} \right\},\quad f_0 = \left\{ \begin{array}{l} x,x,x,x,0,x,x,0,1,1 \\ 0,0,x,0,x,x,x,1,1,x \end{array} \right\}$$

$$f_{0,1} = \left\{ \begin{array}{l} 0,0,1,1,x,x,0,x,x,x \\ 0,x,0,1,x,x,x,0,x,x \\ 0,x,x,0,1,x,x,x,x,0 \end{array} \right\}$$

and the updated sets are $A_0 = \{1,2,4,5\}$, $A_{0,1} = \{1,2,3,4\}$, $A_1 = \{1,2,3,4\}$, $AR_0 = \{1,1,0,1,1,0\}$, $AR_1 = \{3,2,2,1,0,0\}$, $AR_{0,1} = \{3,1,1,1,0,0\}$. We have $A_1 \cap A_0 = \{1,2,4\}$, $A_1 \cap A_0 - A_{0,1} = \emptyset$, so we select the attribute $A_4$, because $4 \in A_1 \cap A_0 \cap A_{0,1}$ and $AR_1[4]AR_{0,1}[4] = 1$.

Apart from that, $f_1^c = \{[3],1,x,1,1,x,x,x,x,0,x\}$, $f_0^c = \{0,0,x,0,x,x,x,1,1,x\}$, $f_{0,1}^c = \{0,x,x,0,1,x,x,x,x,0\}$, $f_1^c \bullet f_0^c = \{x,0,x,x,x,x,x,1,0,x\}$ (a new row of $f_0$), $f_1^c \bullet f_{0,1}^c = \{x,x,x,x,1,x,x,x,0,0\}$. The last result will be appended to table $f_1$ and receive serial no. 4. Consequently,

$$f_1 = \left\{ \begin{array}{l} [1],1,1,x,x,x,x,0,x,x,x \\ [2],1,1,1,x,x,x,x,0,x,x \\ [3],1,x,1,x,x,x,x,x,0,x \\ [4],x,x,x,x,1,x,x,x,0,0 \end{array} \right\}$$

$$f_0 = \left\{ \begin{array}{l} x, x, x, x, 0, x, x, 0, 1, 1 \\ x, 0, x, x, x, x, x, 1, 0, x \end{array} \right\}, \quad f_{0,1} = \left\{ \begin{array}{l} 0, 0, 1, 1, x, x, 0, x, x, x \\ 0, x, 0, 1, x, x, x, 0, x, x \end{array} \right\}$$

and the updated sets are $A_0 = \{2, 5\}$, $A_{0,1} = \{1, 2, 3\}$, $A_1 = \{1, 2, 3, 5\}$, $AR_0 = \{0, 1, 0, 0, 1, 0\}$, $AR_1 = \{3, 2, 2, 0, 1, 0\}$, $AR_{0,1} = \{3, 1, 1, 1, 0, 0\}$. Because $A_1 \cap A_0 = \{2, 5\}$ and $A_1 \cap A_0 - A_{0,1} = \{5\}$, we select $A_5$ but the result of the only multiplication $\{x, x, x, x, 1, x, x, x, 0, 0\} \bullet \{x, x, x, x, 0, x, x, 0, 1, 1\}$ is null. We are left with the initial rows of

$$f_1 = \left\{ \begin{array}{l} [1], 1, 1, x, x, x, x, 0, x, x, x \\ [2], 1, 1, 1, x, x, x, x, 0, x, x \\ [3], 1, x, 1, x, x, x, x, x, 0, x \end{array} \right\}$$

a single row $f_0 = \{x, 0, x, x, x, x, x, 1, 0, x\}$, and $f_{0,1} = \left\{ \begin{array}{l} 0, 0, 1, 1, x, x, 0, x, x, x \\ 0, x, 0, 1, x, x, x, 0, x, x \end{array} \right\}$.
The new sets are $A_0 = \{2\}$, $A_{0,1} = \{1, 2, 3\}$, $A_1 = \{1, 2, 3\}$, $AR_0 = \{0, 1, 0, 0, 0, 0\}$, $AR_1 = \{3, 2, 2, 0, 0, 0\}$, $AR_{0,1} = \{2, 1, 1, 0, 0, 0\}$. We have $A_1 \cap A_0 = \{2\}$ and $A_1 \cap A_0 - A_{0,1} = \emptyset$, so we must select $A_2$. The first two rows of $f_1$ form $f_1^c$ and produce $A_7A_9 \rightarrow A_8$ and a null row. The set $f_0$ is empty and the algorithm terminates (no multiplication $f_1 \bullet f_{0,1}$ is required) giving

$$\pi_X(F) = \{A_8 \rightarrow A_7,\ A_7A_9 \rightarrow A_8\}$$

## 6. Concluding Remarks

The paper presents a solution to an interesting problem how to compute projected functional dependencies effectively without computing a closure. It gives yet another application of Boolean functions in the dependency theory by finding equivalence between projections of FD's and decompositions of Boolean functions. The finding of an algebraic expression for $\pi_X(F)$ seems to be the main contribution of the paper. To make this expression computable, a special tabular representation of FD's has been devised. The partitioning $\{f_0, f_1, f_{0,1}\}$ of the set $F$ is the first step towards an organized way of performing multiplication of rows. Which rows are to be multiplied in each step is primarily determined by the contents of the sets $A_0, A_{0,1}, A_1$ which are updated after each multiplication. To avoid unnecessary multiplications leading to redundant results, the sets $S_1$ are introduced. While these two optimization techniques are essential and do not seem to affect adversely the overall complexity, the need for arrays $AR$ may be questioned, and we leave this issue open. A further study is needed to pick up cases that lead to the generation of redundant dependencies causing exponential complexity. Also, an informal description of the algorithm must be presented in some formal language. These issues will be discussed in a paper which is being prepared. The main contribution of the paper seems to be the application of the concept of a Boolean function to the problem of projected FD's which still awaits a satisfactory algorithmic solution. It seems tempting to pursue a conceptually simple idea ilustrated in Example 2 in order to find a more elegant and efficient method of generating $\pi_X(F)$.

## References


Armstrong W.W. (1974): *Dependency structures of database relationships.* — Proc. 1974 IFIP Congress, Amsterdam: North-Holland, pp.580–583.

Beeri C. and Vardi M.Y. (1981): *The implication problem for data dependencies.* — Lecture Notes in Computer Science 115, New York: Springer-Verlag, pp.73–85.

Berman J. and Blok W.J. (1985): *Positive Boolean dependencies.* — Res. Report, Univ. of Illinois, CS, No.5.

Demetrovics J., Rónyai L. and Hua nam Son (1991): *On the representation of dependencies by propositional logic.* — Lecture Notes in Computer Science, Math. Foundations of Database Systems 91, Heidelberg: Springer-Verlag, pp.230–242.

Fagin R. (1977): *Functional dependencies in a relational database and propositional logic.* — IBM J. Research and Development, v.21, pp.534–544.

Fagin R. (1982): *Horn clauses and database dependencies.* — J. ACM, v.29, No.4, pp.952–985.

Fagin R. and Vardi M.Y. (1986): *The theory of data dependencies-a survey*, In: Mathematics of Information Processing (M. Anshel and W. Gewirtz, Eds.). — Proc. Symp. in Applied Mathematics 34, American Mathematical Society, pp.19–72.

Gottlob G. (1987): *Computing covers for embedded functional dependencies.* — Proc. 6th ACM Symp. *Principles of Database Systems*, pp.58–69.

Gottlob G. (1987): *On the size of nonredumdant FD-covers.* — Information Processing Letters, v.24, No.3, pp.355–360.

Nicolas J.M. (1978): *First order logic formalizations for functional, multivalued and mutual dependencies.* — Proc. ACM-SIGMOD Int. Conf. *Management of Data*, pp.40–46.

Sagiv Y., Delobel C., Parker D.S.JR and Fagin R. (1981): *An equivalence between relational database dependencies and a fragment of propositional logic.* — JACM, v.28, No.3, pp.435–453.

Thalheim B. (1987a): *Design tools for large relational database systems.* — Design Tools for Large Relational Database Systems, MFDBS 87, Berlin, Heidelberg: Springer-Verlag, pp.210–224.

Thalheim B. (1987b): *Dependencies in Relational Databases.* — Leipzig: Teubner-Verlag.

Ullman J.D. (1989): *Principles of database and knowledge-base systems.* — Computer Science Press, v.I, pp.376–445.

Zawadzki W. (1995): *The complexity of computation of projected functional dependencies.* — (in preparation).


Received: November 10, 1994
Revised: October 21, 1995