Theoretical Computer Science 77 (1990) 291-308
North-Holland



# EFFICIENT AND OPTIMAL QUERY ANSWERING ON INDEPENDENT SCHEMES

Paolo ATZENI*
*IASI-CNR, Viale Manzoni 30, 00185 Roma, Italy*

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

Communicated by G. Ausiello
Received February 1988
Revised August 1988

**Abstract.** The total projections of the representative instance have recently been proposed as the basis to generate answers for queries in systems using the Universal Relation Interface. To generate the answers inexpensively, we need to know how to compute the total projections efficiently. For the class of independent schemes, we derive an efficient algorithm that generates optimal unions of simple chase join expressions that compute the total projections with respect to a set of embedded functional dependencies.

## 1. Introduction

During the design or restructuring of relational databases, it is frequent to decompose relations, that is, to replace one relation, with two or more. However, it is useful to offer to the user the possibility of querying the original relations, ignoring its decomposition. The *weak instance model* [13, 19, 20, 18] responds to this need, by allowing us to consider, in a single framework, databases composed of more than one relation. Specifically, with respect to query answering, it allows the formulation of queries on a single relation, and producing answers from data in the decomposed database.

The model can be used as the basis of a form of user interface, called the *Universal Relation Interface*, which presents the database as if it were composed of a single relation, thus relieving the user from specifying logical access paths and connections among the actual relations (see, for example, [17, 24] for surveys, and [4, 15, 23] for discussions).

Queries are posed on a relation defined on all the attributes in the various relations, but not actually stored in the database. The connection between the original (or

* Present address: Dipartimento di Informatica e Sistemistica, Università degli Studi di Napoli, with a cross appointment at IASI, where mail should be addressed.

universal) relation and the actual relations is provided by the representative instance, a relation over the universe $U$ of the attributes, defined, for each database state $r$, as follows. First, a relation over $U$ (called the *state tableau* for $r$, denoted by $T_r$) is formed by taking the union of all the relations in $r$ extended to $U$ by means of distinct variables (or nulls). Then, the chase procedure [16] is applied to $T_r$ to equate symbols and generate new tuples. The chase process essentially performs inferences on data using the given constraints. If a contradiction is found during the chase process, then the representative instance is assumed to be empty. Assuming that a "piece" of information is a "piece" of known information, given any query $Q$ invoiving a set of attributes $X$ that is a subset of the universe $U$, it is meaningful to consider, as the answer to such a query $Q$, the set of tuples in the representative instance that have only constants as values for the attributes in $X$. This set of tuples is called the $X$-total projection of the representative instance. It was shown that the $X$-total projection corresponds to the set of sentences that is logically implied by the database state and the constraints [18]. In this sense, the answer generated by this method is correct.

**Example 1.1.** Consider the database scheme $\mathbf{R} = \{R_1(Course, Tutor, Instructor, Department), R_2(Course, Tutor, Room, Department)\}$, $F = \{Course \rightarrow Instructor, Course \rightarrow Department\}$, and the database state $r: r_1 = \{\langle c_1, t_1, i_1, d_1\rangle\}$, $r_2 = \{\langle c_1, t_2, r_1, d_1\rangle\}$.

Suppose we want to know who are the instructor and tutors of a course. The window on *Course_Instructor_Tutor* is required. In the total projection approach, we first set up the state tableau for $r$. The state tableau $T_r$ is defined on $U$ with two tuples in it. The two tuples are the tuples in the state extending to $U$ with nulls. Since the two tuples in $T_r$ agree on the *Course*-component, and from the functional dependency $Course \rightarrow Instructor$, we can infer that the tuple from $r_2$ has the *Instructor*-component equal to $i_1$. Since no other inferences can be made from the state and the constraints, the corresponding representative instance has the following two tuples: $\{\langle c_1, t_1, i_1, d_1, \phi_1\rangle, \langle c_1, t_2, i_1, d_1, r_1\rangle\}$, where $\phi_1$ is assumed to be a null value. Hence the window on *Course_Instructor_Tutor* is $\{\langle c_1, i_1, t_1\rangle, \langle c_1, i_1, t_2\rangle\}$.

In this approach, the most straightforward way of finding an $X$-total projection is to generate the representative instance and then find the $X$-total projection from it. However, this takes time and space proportional to the size of the database state even if functional dependencies are given and this might take exponential time and space if the full join dependency $\bowtie\mathbf{R}$ is given as a constraint. If the constraints are a set of functional dependencies plus a full acyclic join dependency $\bowtie\mathbf{R}$, then chasing the representative instance can be done in polynomial time proportional to the size of the database state [26]. Sagiv [19, 20] considered a class of database schemes and showed that, for this class, $X$-total projections can be computed using unions of extension joins [12], computable in time polynomial in the size of the database scheme.

Sagiv's method is an interesting approach. Generating the representative instance may be too costly, since it takes time and space polynomial proportional to the size of the database state and answering a query often only involves a very small part of the state. Furthermore, existing systems do not provide facilities to support the chase procedure.

In this paper, we find relational expressions to compute the $X$-total projections of the representative instance when functional dependencies are given as constraints. This class of expressions provides a tool to simulate the representative instance without physically constructing the tableau. For instance in Example 1.1, we can find out the instructor of a course in $r_2$ by joining $R_2$ with $\pi_{Course,Instructor}(R_1)$, since from the functional dependency *Course* → *Instructor*, we know each course has a unique instructor. Therefore, by a simple analysis, the *Course_Instructor_Tutor*-total projection can be computed by the following expression:

$$\pi_{Course,Instructor,Tutor}(R_1) \cup \pi_{Course,Instructor,Tutor}(R_2 \bowtie \pi_{Course,Instructor}(R_1)).$$

In fact, this expression is a union of simple chase join expressions. The optimization of this class of expressions is studied in [3]. To show that simple chase join expressions are a natural way to simulate the representative instance, we derive an efficient algorithm that generates optimal unions of simple chase join expressions which compute the total projections with respect to a set of functional dependencies when an independent scheme is assumed. This generalizes the results obtained by Sagiv [19, 20].

Having defined the necessary notation in Section 2, in Section 3, we give a brief discussion on independent schemes which is essential to the subsequent sections. In Section 4, we show that unions of simple chase join expressions are sufficient to simulate the total projections with respect to a set of functional dependencies when an independent scheme is assumed. And in Section 5, we derive an efficient algorithm that generates the required unions of simple chase join expressions for the total projections. Together with the results obtained in [3], the expressions generated are in some sense minimal in the number of join operations. Finally in Section 6, we conclude with a summary of our contributions.

## 2. Definitions and notation

### *2.1. Basics*

We fix a finite set of attributes $U = \{A_1, \ldots, A_n\}$ and call it the *universe*. A *relation scheme* $R$ is a subset of $U$. A *database scheme* $\mathbf{R} = \{R_1, \ldots, R_k\}$ is a collection of relation schemes such that the union of the $R_i$s is $U$. Associated with each attribute $A_i \in U$ is a set of constants called the *domain* of $A_i$ or $\mathrm{dom}(A_i)$. A *tuple* $t$ over $R_i = \{A_1, \ldots, A_m\}$ is an element of $\mathrm{dom}(A_1) \times \cdots \times \mathrm{dom}(A_m)$. A *relation* $r_i$ over $R_i$ is a set of tuples over $R_i$. A *database state* for a database scheme $\mathbf{R}$ is a function

$r$ that maps every relation scheme $R_i$ in $\mathbf{R}$ to a relation on $R_i$; we write $r = \langle r_1, \ldots, r_k \rangle = \langle r(R_1), \ldots, r(R_k) \rangle$.

We shall consider relational expressions in which the only operators are *select* ($\sigma$), *project* ($\pi$), (*natural*) *join* ($\bowtie$) and *union* ($\cup$). If only the operators select, project and join are involved, the expressions are called *SPJ-expressions.* In the subsequent discussion, the operands of a relational expression are relation schemes in a database scheme.

### 2.2. Tableaux and containment mappings

A *tableau* consists of a *body* and a possibly empty *summary row.* The body of a tableau is a relation over $U' = U \cup \{TAG\}$. Each tuple in the body is simply called a *row.* The tableau domain of $A_i \in U$, $\text{tdom}(A_i)$, is the disjoint union of $\text{dom}(A_i)$, the singleton set $\{a_i\}$, where $a_i$ is called the *distinguished variable* (dv) for $A_i$, and a countable set $\text{Ndv}(A_i)$ of *nondistinguished variables* (ndvs) for $A_i$. The tag domain $\text{tdom}(TAG) = \mathbf{R}$. The elements of $\text{tdom}(A_i)$, for all $A_i \in U$, are ordered by a partial order $<$ such that

- all elements of $\text{dom}(A_i)$ are pairwise incomparable,
- $c < v$, for $c \in \text{dom}(A_i)$, $v \in \text{tdom}(A_i) - \text{dom}(A_i)$,
- $a < b$, for $a$ the dv for $A_i$, $b \in \text{Ndv}(A_i)$,
- $\text{Ndv}(A_i)$ is a linear order set under $<$.

The summary row of a tableau is a tuple over a subset of $U$ called the *target relation scheme.* Where it is defined, the summary row contains only dvs and constants that appear in the body of the tableau.

Let $\nu$ be a *valuation function* that takes $\text{tdom}(A)$ to itself, for each $A \in U'$. Furthermore, for each $A \in U$, $c \in \text{tdom}(A)$, $\nu(c) \leq c$ and $\nu$ is the identity mapping on $\text{tdom}(TAG)$. That is, $\nu$ is a tag-preserving valuation function. Unless otherwise stated, all valuation functions are tag-preserving. A valuation function $\nu$ will also extend to set- and tuple-wise. Let $\{t, t_1, \ldots, t_k\}$ be a set of tuples. We define $\nu(t) = \langle \nu(t[A_1]), \ldots, \nu(t[A_n]) \rangle$ and $\nu(\{t_1, \ldots, t_k\}) = \{\nu(t_1), \ldots, \nu(t_k)\}$. A *containment mapping* $\nu$ from a tableau $T_1$ to another tableau $T_2$ is a valuation function on the set of symbols in rows of $T_1$ to those in rows of $T_2$ such that $\nu(T_1) \subseteq T_2$.

Given tableaux $T$ and $I$ defined on $U'$, $T$ has a nonempty summary row and $I$ may have an empty summary row. The tableau $T$ defines a mapping from $I$ to a relation $T(I)$ defined on the target relation scheme of $T$. $T(I)$ is determined by a set of containment mappings and is defined as follows:

$$T(I) = \{\nu(s) \mid s \text{ is the summary row for } T \text{ and } \nu: T \to I \text{ is a containment mapping from } T \text{ to } I\}.$$

A tableau $T_1$ is said to *contain* $T_2$, written $T_1 \supseteq T_2$, if for every tableau $\sigma$, $T_1(\sigma) \supseteq T_2(\sigma)$. $T_1$ is *equivalent* to $T_2$, denoted by $T_1 \equiv T_2$, if $T_1 \supseteq T_2$ and $T_2 \supseteq T_1$. It is known that $T_1 \supseteq T_2$ if and only if they define the same target relation scheme and there is a containment mapping from $T_1$ to $T_2$ [2].

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

- $u[R_i] = t$;
- $u[A] = b_{ij}$, $b_{ij}$ is a ndv or *null symbol* that appears nowhere else in $T_r$, $A \in U - R_i$;
- $u[TAG] = R_i$.

The summary of $T_r$ is empty.

### 2.3. *Functional dependencies, transformations and chasing*

Unless otherwise stated, the kind of constraints considered here are *functional dependencies* (fds) [8]. Given a set of fds, there are additional dependencies implied by this set. The set of dependencies that are logically implied by $F$ is the *closure* of $F$, denoted by $F^+$. $F$ is said to be *nonredundant* if there is no $X \to A \in F$ such that $(F - \{X \to A\})^+ = F^+$. Given a set of attributes $X$, the *closure* of $X$ with respect to $F$, denoted by $X^+$, is the set of attributes $\{A \mid X \to A \in F^+\}$.

An fd $X \to A$ is said to be *embedded* in a relation scheme $R$ if $XA \subseteq R$. The projection of a set of fds $F$ onto $R_i$, denoted by $F^+ \mid R_i$, is the set of *projected* fds $X \to A \in F^+$ such that $X \to A$ is embedded in $R_i$. A database scheme $\mathbf{R}$ is said to be *cover embedding* for a set of fds $F$ if there exists a set of fds $G$ with $G^+ = F^+$ such that for each fd $X \to A \in G$, $X \to A$ is embedded in some $R_i \in \mathbf{R}$. A database scheme is said to be *dependency preserving*, if for any instance $I$ defined on $U$, $I$ satisfies $F$ implies $\bowtie \pi_{\mathbf{R}}(I)$ also satisfies $F$. By a theorem in [5], $\mathbf{R}$ is cover embedding implies $\mathbf{R}$ is dependency preserving.

Let $F$ be a set of fds. We use $\tau = \langle l_s, l_t, X \to A \rangle$ to represent a *transformation* applying to a tableau $T$, where $l_s$ and $l_t$ are rows in $T$ and $X \to A \in F^+$. A transformation $\tau$ is valid if $l_s[X] = l_t[X]$. A transformation $\tau$ is *applied* to a tableau $T$, denoted by $\tau(T)$, if $\tau$ is valid and $T$ is changed according to the following ways:

- If $l_s[A]$ and $l_t[A]$ are not distinct constants, then replace the higher entry with the other according to the partial order $<$. If both are distinct constants, then $\tau(T) = \emptyset$ and $T$ is said to *contradict* $\tau$.

Let $\tau = \tau_1 \ldots \tau_p$ be a sequence of transformations. Then $\tau(T) = \tau_p(\tau_{p-1}(\ldots \tau_1(T) \ldots))$. Let $\Sigma$ be a set of dependency constraints. The *representative instance* for a state $r$, denoted by $CHASE_\Sigma(T_r)$, is the final nonempty tableau obtained from $T_r$ by applying all valid transformations corresponding to $\Sigma$ exhaustively to $T_r$. The process of applying all transformations to a tableau $T_r$ is called the *chasing* of $T_r$. If no contradiction is found during the chasing of $T_r$, then $r$ is said to be a *(globally) consistent* state [10, 13, 25]. Let $\Sigma$ be a set of constraints. The set of all consistent states for a database scheme $\mathbf{R}$ with respect to $\Sigma$ is denoted by $WSAT(\mathbf{R}, \Sigma) = \{r \mid r \text{ is a state of } \mathbf{R} \text{ and is consistent with } \Sigma\}$. A relation $r_i$ is *consistent* with $\Sigma^+ \mid R_i$ if there is a universal relation $I$ satisfying $\Sigma$ such that $\pi_{R_i}(I) \supseteq r_i$. The *locally consistent states* of $\mathbf{R}$ are elements of the set $LSAT(\mathbf{R}, \Sigma) = \{r \mid r_i \text{ is consistent with } \Sigma^+ \mid R_i, \text{ for each } r_i \in r\}$.

Let $T_r$ be the state tableau for a state $r$. Let $t$ be a tuple in $T_r$ and $X$ be a subset of $U$. We say $t[X]$ is *total* if $t[A_i] \in \mathrm{dom}(A_i)$, for all $A_i \in X$. Let $\pi^{\downarrow}$ be the *total projection operator* which is defined as $\pi^{\downarrow}_X(T_r) = \{t[X] \mid t \in T_r \text{ and } t[X] \text{ is total}\}$. Given a representative instance $CHASE_\Sigma(T_r)$, the *X-total projection* on $CHASE_\Sigma(T_r)$ is defined as $\pi^{\downarrow}_X(CHASE_\Sigma(T_r))$. A database scheme is said to be *bounded* with respect to a set of dependencies if for any $X \subseteq U$ and for any tuple $t$ in the $X$-total projection of the representative instance for any consistent state $r$, $t$ can be derived from $T_r$ by at most $k$ applications of transformation, for some constant $k$. It has been shown that $\mathbf{R}$ is bounded exactly when every $X$-total projection of the representative instance can be computed by a predetermined relational expression [9, 18].

### 2.4. Equivalence, containment and optimality of expressions

Let $E$ be a relational expression with operands in $\mathbf{R} = \{R_1, \ldots, R_k\}$. Then $E(r)$ denotes the value returned by $E$ if a database state $r = \langle r_1, \ldots, r_k \rangle$ on $\mathbf{R}$ is substituted into the corresponding relation variables in $E$ and is evaluated according to the usual definition. Let $E_1$ and $E_2$ be two relational expressions with operands defined on $\mathbf{R}$. $E_1$ is said to *contain* $E_2$, denoted by $E_1 \supseteq E_2$, if for all consistent states $r$ on $\mathbf{R}$, $E_1(r) \supseteq E_2(r)$. $E_1$ is said to be *equivalent* to $E_2$, denoted by $E_1 \equiv E_2$, if $E_1 \supseteq E_2$ and $E_2 \supseteq E_1$. Let $E$ be a union of SPJ-expressions. Then we can always construct a union of tagged tableau $T_E$ for $E$ [22]. Furthermore $T_E(T_r) = E(r)$, for any state $r$ [22]. Let $E$ be a union of SPJ-expressions. Then $E$ is *optimal* (or *minimal*) if there does not exist another equivalent union of SPJ-expressions with a fewer number of join operations.

### 2.5. Derivation sequences and chase join expressions

Given a set of fds $F$, a *derivation sequence* (ds) of some relation scheme $R_i$ is a finite sequence of fds $\langle f_1 : Y_1 \to Z_1, \ldots, f_m : Y_m \to Z_m \rangle$ satisfying the following conditions:

- $Y_j \to Z_j \in F^+$, for all $1 \leq j \leq m$.
- $Y_j \subseteq (\bigcup_{k=1}^{j-1} Y_k Z_k \cup R_i)$ and $Z_j \cap (\bigcup_{k=1}^{j-1} Y_k Z_k \cup R_i) = \emptyset$, for all $1 \leq j \leq m$.

The ds is said to have a *length* of $m$. Essentially a ds of $R_i$ is a sequence of fds used in computing the closure of $R_i$. A ds of $R_i$ *covering* $X$ is a ds of $R_i$ such that $(\bigcup_{j=1}^{m} Y_j Z_j \cup R_i) \supseteq X$.

A ds is said to be *decomposed* if for every fd $Y_i \to Z_i$ in the sequence, $Z_i = A_j$, for some $A_j \in U$. We can always obtain a decomposed ds from a nondecomposed one by replacing $Y_i \to Z_i$ with $Y_i \to A_{i_1}, \ldots, Y_i \to A_{i_p}$, where $Z_i = A_{i_1} \ldots A_{i_p}$, for all $Y_i \to Z_i$ in the ds. Two decomposed dss of $R_i$ are *equivalent* if they are identical up to permutation of fds in the sequences.

Let $R_1$ and $R_2$ be a pair of relation schemes for which there exists $Y \subseteq R_2 - R_1$ such that $R_1 \cap R_2 \twoheadrightarrow Y \in F^+$. The join of $r_1$ and $\pi_{(R_1 \cap R_2) \cup Y}(r_2)$ is called an *extension join* [12]. A generalization of extension joins is called *chase join expressions* (cjes) and is recursively defined as follows [6]:

(1) $R_{i_0}$ is a cje defined on $R_{i_0}$, for any $R_{i_0} \in \mathbf{R}$;

(2) Let $E_i$ and $E_j$ be two cjes defined on $R$ and $S$, respectively. Then $E_i \bowtie \pi_{XY}(E_j)$ is a cje defined on $R \cup Y$, where $X \subseteq R \cap S$, $Y \subseteq S - R$, and $X \to Y \in F^+$.

Given a ds $\langle Y_1 \to Z_1, \ldots, Y_m \to Z_m \rangle$ of a relation scheme $R_{i_0}$ covering $X$, if each fd is embedded in some cje, then we define the *cje E for the ds* as follows:

$$E = \pi_X(R_{i_0} \bowtie \pi_{Y_1 Z_1}(E_{i_1}) \bowtie \cdots \bowtie \pi_{Y_m Z_m}(E_{i_m})),$$

where for each $1 \leq j \leq m$, $Y_j Z_j$ is embedded in $E_{i_j}$, for some cje $E_{i_j}$. In particular, if each $E_{i_j}$ is some relation scheme $R_{i_j}$, $1 \leq j \leq m$, then the cje for the ds is *simple*. In the subsequent discussion, we use simple chase join expression (scje) to mean scje for some ds of a relation scheme covering $X$, for some $X \subseteq U$.

**Example 2.2.** $\mathbf{R} = \{R_1(AB), R_2(ABCDEF)\}$. $F = \{AB \to D, BC \to E, B \to C, D \to F, E \to F\}$. Then the following are dss:

(1) $\langle B \to C, BC \to E, E \to F \rangle$ is a ds of $R_1$ covering $ABCF$;

(2) $\langle AB \to D, D \to F, B \to C \rangle$ is a ds of $R_1$ covering $ABCF$;

(3) $\langle\,\rangle$: the empty sequence is a ds of $R_2$ covering $ABCF$.

The following are scjes for the above three dss, respectively:

(1) $E_1 = \pi_{ABCF}(R_1 \bowtie \pi_{BC}(R_2) \bowtie \pi_{BCE}(R_2) \bowtie \pi_{EF}(R_2))$;

(2) $E_2 = \pi_{ABCF}(R_1 \bowtie \pi_{ABD}(R_2) \bowtie \pi_{DF}(R_2) \bowtie \pi_{BC}(R_2))$;

(3) $E_3 = \pi_{ABCF}(R_2)$.

## 3. Independent schemes

A scheme $\mathbf{R}$ is *independent* with respect to a set of constraints $\Sigma$ if $WSAT(\mathbf{R}, \Sigma) = LSAT(\mathbf{R}, \Sigma)$. Sagiv [19, 20] characterized a class of independent schemes when the constraints are a set of key dependencies. Graham and Yannakakis [11], and independently Ito et al. [14], generalized Sagiv's independence and presented a number of results on independence, including a polynomial time test for independence with respect to a set of embedded fds.

A database scheme $\mathbf{R}$ is independent with respect to $\Sigma = H \cup \{\bowtie \mathbf{R}\}$, where $H$ is a set of fds, if verifying that each relation is consistent with its projected dependencies suffices to ensure the state is globally consistent. In fact, the set of projected dependencies is a set of fds.

**Theorem 3.1** (Graham and Yannakakis [11]). *Let $\Sigma = H \cup \{\bowtie \mathbf{R}\}$ and $G$ be the set of fds implied by $\Sigma$. The database scheme $\mathbf{R}$ is independent with respect to $\Sigma$ if and only if the following two conditions are satisfied*:

(1) $\mathbf{R}$ *embeds a cover $F$ of $G$*;

(2) $\mathbf{R}$ *is independent with respect to $F$ alone.*

Graham and Yannakakis derived an efficient algorithm which finds the cover $F$ if $\mathbf{R}$ is independent with respect to $\Sigma$ and $\Sigma$ is logically equivalent to $F \cup \{\bowtie \mathbf{R}\}$.

In the subsequent discussion we shall assume the embedded cover is given when an independent scheme is considered. They also found a polynomial time algorithm which recognizes exactly the class of independent schemes when an embedded cover of fds is considered. In the remainder of this section we will informally describe the recognition algorithm (in the following called *Algorithm GY*) and present some properties of independent schemes which will be used later.

A scheme is not independent when, chasing the tableau for a locally consistent state, a contradiction arises. That is, the chase tries to equate two distinct constants. Intuitively, this means that there are two different ways to derive some fd in $F^+$. We say $X$ is a *left-hand side* (lhs) from $R_k$ if $X \to A \in F_k$, for some $A$. Since the same set of attributes can be lhs from two or more relation schemes, we distinguish the appearances of the same set of attributes as distinct lhss in distinct schemes. Algorithm GY tests independence by building for each fd (of the form $R_j \to A$) in $F^+$ a "unique" minimal derivation. The algorithm computes the closure of each relation scheme $R_j \in \mathbf{R}$ with respect to $F$, associating a tableau with each attribute in $R_j^+$ and with each lhs $X \subseteq R_j^+$ from some $R_i$. During the computation of $R_j^+$, an attribute $A$ is said to be *available* after it has been discovered that $A \in R_j^+$. A lhs $X$ is available when each $A \in X$ is available. The tableau for an attribute $A \in R_j^+$ is indicated with $T_j(A)$ and is defined when $A$ becomes available; the tableau for an (available) lhs $X$ from $R_i$ is defined as the union of the tableaux $T_j(A)$, for $A \in X$, plus a row ($X^*$-row in the following) with dvs for the attributes in $X^*$ (where $X^*$ denotes the local closure of $X$, that is, its closure with respect to $F_i$), unique ndvs for the other attributes, and tag $R_i$. The tableau for $X$ is denoted by $T_j(X)$. Initially, the set of the available attributes is initialized to $R_j$, and the tableau for each $A \in R_j$ is the empty tableau $\emptyset$. A lhs $X$ is *weaker* than another lhs $Y$, denoted by $X \leqslant Y$, if $T_j(X) \supseteq T_j(Y)$.[1] $X \equiv Y$ abbreviates $X \leqslant Y$ and $Y \leqslant X$. The algorithm distinguishes the available lhss into *unprocessed* and *processed*, and processes lhss in the order of weakness and halts either when it discovers a contradiction (we omit details) or when all the available lhss have been processed. At each iteration, a weakest lhs $X$ from some $R_i$ is *picked*, and all the attributes in its local closure $X^*$ (with respect to $F_i$) that are not yet available (let us indicate this set with $X^*_{\text{new}}$) become available, and for each of them the tableau is defined to be equal to the tableau of $X$. The following is the algorithm for testing independence.

**Algorithm GY** (*Testing for independence*)
*Input*: A database scheme $\mathbf{R}$ and a set of embedded fds $F$.
*Output*: Reject or accept $\mathbf{R}$.
*Method*: For each $R_j \in \mathbf{R}$ do:
{Initialization}
Mark *available* the attributes in $R_j$, *not available* the others.
For each available attribute $A$, define $T_j(A) = \emptyset$.

[1] Graham and Yannakakis [11] used $T(X) \leqslant T(Y)$ instead of $T(X) \supseteq T(Y)$.

Mark *unprocessed* every lhs from $R_i \neq R_j$, for every $i$.
Compute the available lhss, their tableaux, and their order under $\leqslant$.
{The main loop.}
While there is an available but unprocessed lhs $X$ from some $R_i$ do:
  Pick a weakest such $X$ and let $E(X)$ be the set of available lhss from $R_i$ that are equivalent to $X$ under $\leqslant$.
  Determine the set $W(X)$ of the available lhss from $R_i$ strictly weaker than $X$.
  Let $X^*_{\text{old}}$ be the closure of $X$ with respect to the set of fds
    $WF(X) = \{Z \rightarrow Z^* \mid Z \in W(X)$ and $Z^*$ is the closure with respect to $F_i\}$.
  Let $X^*_{\text{new}} = X^* - X^*_{\text{old}}$.
  Verify that each attribute in $X^*_{\text{new}}$ is not available; if one is, *halt and reject.*
  For each $Y$ in $E(X)$ (besides $X$) do:
    Compute the closure $Y^*_{\text{old}}$ with respect to $WF(X)$ and verify that $Y^*_{\text{old}} = X^*_{\text{old}}$; if not, *halt and reject.*
  Mark every attribute $A \in X^*_{\text{new}}$ *available* and define $T_i(A) = T_i(X)$.
  Update the set of available lhss, compute their tableaux and update their order under $\leqslant$.
  Mark *processed*, every unprocessed lhs $Z$ of $R_i$ with $Z^* \subseteq X^*$.

Graham and Yannakakis [11] proved the correctness of the algorithm; we just repeat here a lemma that will be used later.

**Lemma 3.2** (Graham and Yannakakis [11]). *Let $R_j$ be a relation scheme selected at the outer for loop. If $\mathbf{R}$ is independent and $X$ is a processed lhs from some $R_i$ such that $R_j^+ \supseteq X$, and $A \in X^*$, where $X^*$ is the local closure with respect to $F_i$, then $T_j(A) \supseteq T_j(X)$.*

## 4. Computing total projections for independent schemes with unions of scjes

In this section, we show that there is an algorithm which generates unions of scjes to compute the $X$-total projections with respect to an embedded cover $F$ when an independent scheme is considered. In the rest of this section, we derive the algorithm by showing that the representative instance for any legal state defined on an independent scheme can be obtained by chasing the tableau for the state in a particular way. Without loss of generality, we assume $F = \bigcup_i F_i$, where $F_i$ is a nonredundant cover for $F^+ \mid R_i$, for any $R_i \in \mathbf{R}$. Also, all dss are assumed to be decomposed. The following algorithm will convert $T_r$ to $CHASE_F(T_r)$.

**Algorithm 4.1** (*Chasing the tableau for an independent state*).
*Input*: $T_r$, where $r$ is a consistent state defined on an independent scheme $\mathbf{R}$, and a set of fds $F$ as defined above.
*Output*: $CHASE_F(T_r)$.

*Method*:

(1) Finish = false.
Loop until finish = true: If there is an $X \to A \in F$ and $t_1$, $t_2$ in $T_r$ such that $t_1[XA]$ are constants and $t_1[X] = t_2[X]$ and $t_2[A]$ is a null symbol then $t_2[A]$ is equated to $t_1[A]$, else finish = true.

(2) Let the resulting tableau after step (1) be $T_r^*$. Chase $T_r^*$ to get $CHASE_F(T_r)$.

We claim during step (2) of Algorithm 4.1, only null symbols are equated. Before we prove the claim, the following lemma is needed.

**Lemma 4.2.** *Let $t$ be a tuple defined on $U$. Let $r = \pi_{\mathbf{R}}(t)$, where $\mathbf{R}$ is cover embedding for a set of fds $F$. Let $R_{i_0} \in \mathbf{R}$ and $t_{i_0}$ be the tuple in $CHASE_F(T_r)$ corresponding to $\pi_{R_{i_0}}(t)$. Then $t_{i_0}[R_{i_0}^+] = t[R_{i_0}^+]$.*

**Proof.** By assumption, we know each fd in $F$ is embedded in some relation scheme. We prove the lemma by showing inductively that $t_{i_0}[R_{i_0}X_1A_1 \ldots X_kA_k] = t[R_{i_0}X_1A_1 \ldots X_kA_k]$, where $\langle X_1 \to A_1, \ldots, X_k \to A_k \rangle$ is a ds of $R_{i_0}$ covering $A_1A_2 \ldots A_k$.

*Basis*: $k = 0$. When no fd is used then by construction of $T_r$, $t_{i_0}[R_{i_0}] = \pi_{R_{i_0}}(t) = t[R_{i_0}]$. Hence the basis is trivially established.

*Induction*: $k > 0$. Suppose the induction hypothesis is true for all dss of $R_{i_0}$ with length less than $k$. Consider a ds of $R_{i_0}$ with length $k$. Let $\langle X_1 \to A_1, \ldots, X_k \to A_k \rangle$ be the ds and let each $X_k \to A_k$ be embedded in $R_{i_k}$, for some $R_{i_k} \in \mathbf{R}$. By the induction hypothesis, $t_{i_0}[R_{i_0}X_1A_1 \ldots X_{k-1}A_{k-1}] = t[R_{i_0}X_1A_1 \ldots X_{k-1}A_{k-1}]$. Since $X_k \subseteq (\bigcup_{p=1}^{k-1} X_pA_p \cup R_{i_0})$, $t_{i_0}[X_k] = t[X_k]$. Since $t[R_{i_k}] = t_{i_k}[R_{i_k}]$ and $X_k \to A_k$ is embedded in $R_{i_k}$, $t[X_kA_k] = t_{i_k}[X_kA_k]$. Rows $t_{i_k}$ and $t_{i_0}$ are both in the tableau and their $X_k$-components are equal, hence $t_{i_0}[A_k] = t_{i_k}[A_k] = t[A_k]$ in $CHASE_F(T_r)$. Therefore $t_{i_0}[R_{i_0}X_1A_1 \ldots X_kA_k] = t[R_{i_0}X_1A_1 \ldots X_kA_k]$ in $CHASE_F(T_r)$. This completes the induction proof. □

Having shown the lemma above, we are ready to prove our claim.

**Lemma 4.3.** *Let $\mathbf{R}$ be an independent scheme with respect to an embedded cover $F$, where $F$ is a set of fds. Let $r \in WSAT(\mathbf{R}, F)$. Consider chasing $T_r$ using Algorithm* 4.1. *In step* (2) *of the algorithm, only null symbols are equated.*

**Proof.** Assume otherwise. Let $\tau = \tau_1 \ldots \tau_l$ be a sequence of transformations used in step (2) such that $\tau_l = \langle t_1, t_2, X \to A \rangle$ is the first transformation to transform $t_2[A]$ from null to constant. Let $S = \{A_1, \ldots, A_m\}$ such that the $S$-components of $t_2$ are all the null symbols at the stage after $\tau_{l-1}$ is applied to the tableau but before $\tau_l$ is invoked. Suppose $t_2$ originates from $r_p$. Clearly $A \in R_p^+$. Let us define a universal tuple $t$ as follows: $t[B] = n$ where $n$ is a value that appears nowhere else if $B \in S$, and $t[B] = t_2[B]$ otherwise. Let $\pi_{\mathbf{R}}(t) = \{s_1, \ldots, s_k\}$, where $s_i = \pi_{R_i}(t)$, $1 \leq i \leq k$. Let

us add each tuple $s_i$ to each $r_i$ to form a new state $r' = \langle r'_1, \ldots, r'_k \rangle$. We claim that each $r'_i = r_i \cup \{s_i\}$ satisfies $F^+ | R_i$. Suppose it does not, then there exists a tuple $t' \in r_i$ such that $t'$ and $s_i$ violate an fd $V \rightarrow C$. Clearly, $V$ does not contain any attribute in $S$, for if it does $t'[V] \neq s_i[V]$. So, $V \subseteq R_i - S$. Consider the following possible cases for $C$:

*Case* 1: $C \in S$. In this case, $t_2[V] = s_i[V] = t'[V]$ and $t_2[C]$ should be assigned the constant $t'[C]$ in step (1) of Algorithm 4.1. Hence a contradiction.

*Case* 2: $C \notin S$. So $VC \subseteq R_i - S$. This implies $r \notin WSAT(\mathbf{R}, F)$. Again a contradiction. Therefore $r' \in LSAT(\mathbf{R}, F)$.

Let us chase $T_r$, as follows:

(1) Apply step (1) of Algorithm 4.1 to $T_r$.

(2) Apply $\tau - \{\tau_i\}$ to the tableau returned by step (1) above.

(3) Chase the part of the tableau for the $s_i$s tuples. Since $\mathbf{R}$ is cover embedding, by Lemma 4.2, the tuple corresponding to $s_p$ becomes constants exactly in the positions $R_p^+$.

We claim that from the above we can deduce that $r'$ is not a consistent state. In step (2) above, the two tuples $t_1$ and $t_2$ have their $X$-components equal and $t_1[A]$ is a constant. But $t_2[R_p] = s_p[R_p]$, and $A \in R_p^+$, hence $t_2[A] = s_p[A]$ in $CHASE_F(T_r)$. But $A \in S$, hence $s_p[A]$ is a constant that appears nowhere else. Therefore $t_2$ and $t_1$ will violate $X \rightarrow A$. Hence $r' \notin WSAT(\mathbf{R}, F)$. Therefore $\mathbf{R}$ cannot be independent. A contradiction. □

Chan and Mendelzon showed that each derived value in $CHASE_F(T_r)$ is "uniquely" derived from a relation for an independent scheme.

**Lemma 4.4** (Chan and Mendelzon [7]). *Let $\mathbf{R}$ be an independent scheme with respect to an embedded cover $F$, where $F$ is a set of fds. Then for any $r \in WSAT(\mathbf{R}, F)$ and for any nonredundant cover $F_j$ of $F^+ | R_j$, for any $R_j \in \mathbf{R}$ and for any $X \rightarrow A \in F_j$, $\pi^{\downarrow}_{XA}(CHASE_F(T_r)) = \pi_{XA}(r_j)$.*

Let $r \in WSAT(\mathbf{R}, F)$ and let $t \in CHASE_F(T_r)$, where $t$ originates from $r_i$. Define $T = \{A | t[A]$ becomes constant in the chase process$\} = R_i A_1 \ldots A_k$, $k \geq 0$. Using Algorithm 4.1 and by Lemma 4.3, there exists a sequence of transformations $\chi$ that changes $t[A_1 \ldots A_k]$ from nulls to constants. Without loss of generality, we assume $t[A_1 \ldots A_k]$ become constants in the order $A_1, \ldots, A_k$. Clearly each entry $t[A_i]$ is changed by exactly one transformation. Therefore $|\chi| = k$ and let the fds involved in $\chi$ be $\langle X_1 \rightarrow A_1, \ldots, X_k \rightarrow A_k \rangle = \langle f_1, \ldots, f_k \rangle$, $k \geq 0$. By the assumption of $F$, each $f_j$ is embedded in some relation scheme. In an independent scheme, each $f_j$ is embedded in at most one relation scheme. Hence each $f_i$ is embedded in exactly one relation scheme, let the relation scheme be $R_{i_j}$.

It is clear that $\langle f_1, \ldots, f_k \rangle$ is a ds of $R_i$ covering $\{A_1, \ldots, A_k\}$. This follows directly from step (1) of Algorithm 4.1. We claim that the following scje $E$ for the ds

$\langle f_1, \ldots, f_k \rangle$ produces a relation containing $t[T]$,

$$E = R_i \bowtie \pi_{X_1 A_1}(R_{i_1}) \bowtie \cdots \bowtie \pi_{X_k A_k}(R_{i_k}), \quad \text{where } R_{i_j} \text{ embeds } f_j.$$

**Lemma 4.5.** *Let $t$, $T$ and $E$ be defined above. Then $t[T] \in E(r)$.*

**Proof.** Prove by induction on the number $k$ of transformations that change $t$ during the chase process.

*Basis*: $k = 0$. $T = R_i$. $E = R_i$ and since $t$ originates from $r_i$, $t[T] \in E(r)$. Hence the basis is established.

*Induction*: $k > 0$. Suppose the induction hypothesis is true for all tuples $t \in T_r$ such that $t[R_i A_1 \ldots A_{k-1}]$ are constants after $t$ is changed by $k-1$ transformations in step (1) of Algorithm 4.1. Suppose $t[R_i A_1 \ldots A_k]$ become constants in the chase process. Let $\chi = \langle s, t, X_k \to A_k \rangle$ be the transformation that transforms $t[A_k]$ from null to constant in step (1) of Algorithm 4.1, where $X_k \to A_k$ is embedded in $R_{i_k}$. By the induction hypothesis, $t[R_i A_1 \ldots A_{k-1}]$ is contained in some scje $E$ for a ds of $R_i$, where $E = R_i \bowtie \pi_{X_1 A_1}(R_{i_1}) \bowtie \cdots \bowtie \pi_{X_{k-1} A_{k-1}}(R_{i_{k-1}})$. We want to show that $t[R_i A_1 \ldots A_k]$ is contained in the scje $E \bowtie \pi_{X_k A_k}(R_{i_k})$.

Before $\chi$ is invoked $s[X_k A_k]$ are constants. By the assumption of $F$ and by Lemma 4.4, $s[X_k A_k] \in \pi_{X_k A_k}(r_{i_k})$. Hence $t[R_i A_1 \ldots A_k] \in E \bowtie \pi_{X_k A_k}(r_{i_k})$. This completes the induction proof. □

Next, we want to show that in general any scje for a ds of $R_i$ with only produce total tuples in $CHASE_F(T_r)$. Let a ds of $R_i$ be $\langle X_1 \to A_1, \ldots, X_k \to A_k \rangle$. Let $E = R_i \bowtie \pi_{X_1 A_1}(R_{i_1}) \bowtie \cdots \bowtie \pi_{X_k A_k}(R_{i_k})$ be a scje.

**Lemma 4.6.** *Let $r \in WSAT(\mathbf{R}, F)$, $\mathbf{R}$ is any database scheme. Let $E$ be defined above. For any $t \in E(r)$, $t$ is a total tuple in $CHASE_F(T_r)$.*

**Proof.** Follows from Theorem 5 of [18]. □

By Lemmas 4.5 and 4.6, the set of all scjes for dss of $R_i$s produces exactly the set of total tuples in the representative instance. Hence we have an algorithm to compute total projections with respect to the embedded cover $F$ for an independent scheme. The method is as follows. First find a cover $F = \bigcup_i F_i$, where $F_i$ is a nonredundant cover for $F^+ | R_i$, for every $R_i \in \mathbf{R}$. By Graham and Yannakakis's algorithm [11], a cover of $F^+ | R_i$ can be found in polynomial time and hence finding the nonredundant covers $F_i$s can be done in polynomial time. Then for each $R_i \in \mathbf{R}$ such that $R_i^+ \supseteq X$, find all nonequivalent dss of $R_i$ covering $X$. The dss are of the form $\langle X_1 \to A_1, \ldots, X_m \to A_m \rangle$, where $X_j \to A_j \in F$, for all $1 \leq j \leq m$.

**Theorem 4.7.** *Let $\mathbf{R}$ be an independent scheme with respect to an embedded cover $F$ and let $X \subseteq U$. Then the $X$-total projection of the representative instance is given by $E = \bigcup_i \pi_X(E_i)$, where $E$ is the union of scjes obtained as described above.*

**Proof.** Follows from Lemmas 4.5 and 4.6. □

As a consequence of Theorem 4.7, independent schemes are bounded with respect to the embedded cover $F$.

**Corollary 4.8.** *Let* $\mathbf{R}$ *be an independent scheme with respect to an embedded cover* $F$. *Then* $\mathbf{R}$ *is bounded.*[2]

## 5. Efficient generation of union of scjes

In this section we will show that, for every $X \subseteq U$, it is possible to build in polynomial time an optimal expression that computes the $X$-total projection with respect to an embedded cover $F$ when an independent scheme is assumed.

We know from Theorem 4.7 that, for every $X$, the $X$-total projection of the representative instance can be computed as a union of scjes. The main result of this section shows that it is not necessary to consider the scje for every ds covering $X$, but that, for every $R_j$ such that $R_j^+ \supseteq X$, there is one scje that contains all other possible scjes based on $R_j$ involved in the aforementioned union. Intuitively, this scje corresponds to the union of the tableaux generated by Algorithm GY for the attributes $A \in X$. In the remainder of this section, a ds $\tau$ is a ds for $R_j$ covering $X$ means a ds for $R_j$ covering $X$ as defined in Section 4.

**Lemma 5.1.** *Let* $R_{i_0} \in \mathbf{R}$, $A \in R_{i_0}^+$, *and* $\tau$ *be a ds for* $R_{i_0}$ *covering* $A$ *and* $T_{i_0}(A)$ *the tableau for* $A$ *constructed by Algorithm GY when* $R_{i_0}$ *is processed by the outer for loop. For every row* $t \in T_{i_0}(A)$ (*let it be a* $Z^*$*-row with tag* $R_{i_j}$), *there is an fd from* $R_{i_j}$ *in* $\tau$ *with lhs* $Y$ *such that* $Z^* \subseteq Y^*$, *where* $Z^*$ *and* $Y^*$ *are local closures with respect to* $F_{i_j}$.

**Proof.** We assume, without loss of generality, that $\tau$ is a decomposed ds. The proof proceeds by induction on the length $n$ of $\tau = \langle Y_1 \to A_1, \ldots, Y_n \to A_n \rangle$. Let $Y_j \to A_j$ be embedded in $R_{i_j}$, for every $1 \leq j \leq n$.

*Basis*: $n = 0$. Then $A \in R_{i_0}$. Hence $T_{i_0}(A)$ is an empty tableau and therefore the basis is trivially established.

*Induction*: $n > 0$. Let $\tau' = \langle Y_1 \to A_1, \ldots, Y_{n-1} \to A_{n-1} \rangle$, that is, $\tau' = \tau - \langle Y_n \to A_n \rangle$. If $A_n \neq A$, then $\tau'$ is a ds for $R_{i_0}$ covering $A$ of length $n-1$, and so, by the induction hypothesis, the lemma holds. If $A_n = A$, then for every $B \in Y_n$, $\tau'$ is a ds for $R_{i_0}$ covering $B$ with length $m < n$; so, by the induction hypothesis, for each row in $T_{i_0}(B)$ ($Z^*$-row with tag $R_{i_j}$), there is in $\tau'$ an fd from $R_{i_j}$ with lhs $Y$ such that $Z^* \subseteq Y^*$. Since $Y_n \to A_n \in F_{i_n}$ and $Y_n \subseteq R_{i_0}^+$, $T_{i_0}(Y_n)$ will eventually be constructed and is defined as $T_{i_0}(Y_n) = \bigcup_{B \in Y_n} T_{i_0}(B) \cup \{Y_n^*\text{-row}\}$. Since $Y_n \to A_n \in \tau$, for each row in $T_{i_0}(Y_n)$ (let it be a $Z^*$-row with tag $R_{i_j}$), there is an fd on $R_{i_j}$ in $\tau$ with lhs

[2] Independently Maier et al. [17] obtained a similar result.

$Y$ such that $Z^* \subseteq Y^*$. Finally, by Lemma 3.2, $T_{i_0}(A) \supseteq T_{i_0}(Y_n)$ and by the definitions of $T_{i_0}$ and the containment of tableaux, the induction hypothesis also holds for a ds of length $n$. This completes the induction proof. □

This result can be easily extended from ds covering single attribute, tc ds covering sets of attributes.

**Corollary 5.2.** *Let $\tau$ be a ds for $R_{i_0}$ covering $X$. Consider $R_{i_0}$ is processed by the outer for loop in Algorithm GY. Then for every $A \in X$, for every row $t \in T_{i_0}(A)$ ($Z^*$-row with tag $R_{i_j}$), there is an fd from $R_{i_j}$ in $\tau$ with lhs $Y$ such that $Z^* \subseteq Y^*$, where $Z^*$ and $Y^*$ are local closures with respect to $F_{i_j}$.*

**Proof.** Immediately follows from Lemma 5.1, since a ds for $X$ is also a ds for every $A \in X$. □

The previous corollary shows the close relationship existing between $T_{i_0}(X)$ and all the dss for $R_{i_0}$ covering $X$: $T_{i_0}(X)$ is "dominated" by each ds. Now, we show how as a consequence, it is possible to define, from $T_{i_0}(X)$, a "minimal" ds for $R_{i_0}$ covering $X$. First of all, for every $R_{i_0}$, for every $A \in R_{i_0}^+$, we define a ds for $R_{i_0}$ covering $A$. These dss are defined recursively, while running Algorithm GY for $R_{i_0}$, the ds covering $A$ being defined when $A$ becomes available, as follows:

$$\tau^*_{i_0,A} = \begin{cases} \emptyset & \text{if } A \in R_{i_0} \\ \bigcap_{B \in Y} \tau_{i_0,B} \cup \{Y \to Y^*_{\text{new}}\} & \text{where } Y \text{ is the lhs picked when } A \text{ becomes available.} \end{cases}$$

Then, for every $X \subseteq R_{i_0}^+$, we define $\tau^*_{i_0,X} = \bigcup_{A \in X} \tau^*_{i_0,A}$. In both the constructions, we consider the order of fds is meaningful (as it is in dss) and preserved by unions. It is easy to show that the construction is well-defined, that is, $\tau^*_{i_0,X}$ is actually a ds for $R_{i_0}$ covering $X$. Now, we show in this sense $\tau^*_{i_0,X}$ is minimal among the dss for $R_{i_0}$ covering $X$. We need a lemma first.

**Lemma 5.3.** *For every fd $Z \to Z^*_{\text{new}}$ (from $R_{i_j}$) in $\tau^*_{i_0,X}$, there is an attribute $A \in X$ and a row $t \in T_{i_0}(A)$ such that $t$ is a $Z^*$-row with tag $R_{i_j}$.*

**Proof.** By the construction of $\tau^*_{i_0,X}$ and $T_{i_0}(A)$. □

**Theorem 5.4.** *Let $\tau_X$ be a ds for $R_{i_0}$ covering $X$. Then, for every fd $Z \to Z^*_{\text{new}}$ (from $R_{i_j}$) in $\tau^*_{i_0,X}$, there is an fd from $R_{i_j}$ in $\tau_X$ with lhs $Y$ such that $Z^* \subseteq Y^*$, where $Z^*$ and $Y^*$ are closures with respect to $F_{i_j}$.*

**Proof.** Again, we assume $\tau_X$ to be decomposed; if it is not, we can consider the corresponding decomposed ds.

Let $Z \to Z^*_{\text{new}}$ be an fd in $\tau^*_{i_0,X}$ from $R_{i_j}$. From the previous lemma, there is an attribute $A \in X$ and a row $t \in T_{i_0}(A)$, such that $t$ is a $Z^*$-row with tag $R_{i_j}$. Then, by Corollary 5.2, there is an fd from $R_{i_j}$ in $\tau_X$ with lhs $Y$ such that $Z^* \subseteq Y^*$. □

We first state a result from [3] which will be used in the next theorem.

**Theorem 5.5** (Atzeni and Chan [3]). *Let* $\tau = \langle Y_1 \to Z_1, \ldots, Y_m \to Z_m \rangle$ *and* $\chi = \langle V_1 \to W_1, \ldots, V_n \to W_n \rangle$ *be dss for* $R_q$ *covering* $X$. *Let* $E_\tau = \pi_X(R_q \bowtie \cdots \bowtie \pi_{Y_m Z_m}(R_{i_m}))$ *and* $E_\chi = \pi_X(R_q \bowtie \cdots \bowtie \pi_{V_n W_n}(R_{p_n}))$ *be scjes for* $\tau$ *and* $\chi$, *respectively. Then* $E_\tau \supseteq E_\chi$ *if and only if for each* $Y_j \to Z_j$ *in* $\tau$, *there is a* $V_k \to W_k$ *in* $\chi$ *such that* $V_k^+ \supseteq Y_j^+$ *and* $R_{i_j} = R_{p_k}$.

We are finally ready for the main theorem. We will show that, for each $X$, and for each $R_i$ whose closure covers $X$, there is a scje (namely, the scje corresponding to $\tau^*_{i,X}$) that contains all the others. As a consequence, for every $X$, the $X$-total projection of the representative instance can be computed as a union of at most $k$ scjes, where $k$ is the number of relation schemes whose closures contain $X$. Let $E^*_{i_0,X}$ be the scje corresponding to $\tau^*_{i_0,X}$.

**Theorem 5.6.** *Let* $E$ *be a scje for a ds* $\tau$ *of* $R_{i_0}$ *covering* $X$. *Then* $E^*_{i_0,X} \supseteq E$.

**Proof.** First observe that if $\mathbf{R}$ is independent, and $Y$ and $Z$ are lhs of fds on the same relation scheme, then $Y^* \supseteq Z^*$ if and only if $Y^+ \supseteq Z^+$. Then the theorem follows from Theorems 5.4 and 5.5. □

**Corollary 5.7.** *For every* $X$, *the* $X$*-total projection of the representative instance is given by* $\bigcup_{R_i^+ \supseteq X} E^*_{i,X}$.

From the above results, it is possible to derive an algorithm which, for any given $X \subseteq U$, produces in polynomial time an expression $E$ computing the $X$-total projection with respect to an embedded cover $F$. Together with the results obtained in [3], this expression can be optimized efficiently. A version of the algorithm is as follows.

**Algorithm 5.8.** Generate optimal unions of scjes for total projections when independence is assumed.
*Input*: An independent scheme, an embedded cover $F$, $X \neq \emptyset$ and $X \subseteq U$.
*Output*: Optimal expression to compute the $X$-total projection with respect to $F$.
*Method*:

(1) For each $R_i$ such that $R_i^+ \supseteq X$ do: compute $E^*_{i,X}$.
(2) Let $E = \bigcup_{R_i^+ \supseteq X} E^*_{i,X}$.
(3) Minimize the number of join operations in $E$.

Step (1) is performed according to the definitions, running Algorithm GY to build $\tau^*_{i,X}$ and then the corresponding scje, and therefore it has the same complexity as Algorithm GY. Step (3) can be performed efficiently using the results obtained in [3]. By a theorem of Sagiv and Yannakakis [22], the expression obtained is minimal in the number of join operations.

**Corollary 5.9.** *Suppose we are given an independent scheme with respect to an embedded cover F. Then for every $X \subseteq U$, we can build in polynomial time a minimal expression to compute the X-total projection of the representative instance.*

**Example 5.10.** Let us consider a database scheme over the attributes *Driver*, *LicenseNo*, *Model*, *Version*, *Speed*, *Price*, *Country_of_Origin*, *Tax_Rate* and they are abbreviated as their respective first letters in the following. Let $\mathbf{R} = \{R_1(DL), R_2(LMV), R_3(MVC), R_4(MVSP), R_5(CT)\}$. Let $F = \{L \to MV, M \to C, MV \to PS, C \to T\}$.

By means of Algorithm GY, we can verify that $\mathbf{R}$ is independent. Suppose we want the window over *MCS*, i.e., the relationship between models of cars, countries of origin and speed. *MCS* is contained in the closures of $R_1$, $R_2$, $R_3$ and $R_4$. So step (1) of Algorithm 5.8 generates four expressions:

$$E^*_{1,MCS} = R_1 \bowtie \pi_{LMV}(R_2) \bowtie \pi_{MC}(R_3) \bowtie \pi_{MVSP}(R_4);$$

$$E^*_{2,MCS} = R_2 \bowtie \pi_{MC}(R_3) \bowtie \pi_{MVSP}(R_4);$$

$$E^*_{3,MCS} = R_3 \bowtie \pi_{MVSP}(R_4);$$

$$E^*_{4,MCS} = R_4 \bowtie \pi_{MC}(R_3).$$

By results in [3], $E^*_{1,MCS}$, $E^*_{2,MCS}$ and $E^*_{3,MCS}$ are all contained in $E^*_{4,MCS}$. So we are left with $E = E^*_{4,MCS}$. Since all the subexpressions come from different relation schemes, no subexpression can be deleted. Hence the final expression for the computation of the *MCS*-total projection of the representative instance is:

$$E = R_4 \bowtie \pi_{MC}(R_3).$$

Although the expression $E$ generated by our method is used to compute the $X$-total projections with respect to $F$, this method can still be used when $\Sigma = F \cup \{\bowtie \mathbf{R}\}$ is given as a constraint. It was proven in [7] that given a dependency preserving scheme $\mathbf{R}$, for any $R_i \in \mathbf{R}$, and for any $X \subseteq R_i$, $\pi^{\downarrow}_X(CHASE_F(T_r)) = \pi^{\downarrow}_X(CHASE_\Sigma(T_r))$. Since independent schemes are cover embedding and hence dependency preserving, the $X$-total projection with respect to $\Sigma$ can be computed by our method for any $X \subseteq R_i$, and for any $R_i \in \mathbf{R}$. Also, if the independent scheme is a lossless join decomposition [1], it is easy to verify that $\pi^{\downarrow}_X(CHASE_F(T_r)) = \pi^{\downarrow}_X(CHASE_\Sigma(T_r))$, for any $X \subseteq U$. Hence our method is still applicable in these cases.

## 6. Concluding remarks

Using the representative instance as a query-answering device is an interesting approach. Facing these systems is the problem of how to generate answers efficiently. We proposed scjes as an efficient and natural way to simulate the total projections. More specifically, we showed that for any $X \subseteq U$, the $X$-total projection of $CHASE_F(T_r)$ can be computed efficiently by a union of scjes when an independent scheme is assumed.

Independently, several other researchers obtained similar results recently. Ito et al. [14] studied the same problem as we did and they derived an $O(n \times |F| \times \|F\|)$ algorithm that generates relational expressions that compute the total projections for an independent scheme when an embedded cover is assumed. Sagiv [21] studied query evaluation in independent schemes when the set of constraints is the union of the full join dependency $\bowtie \mathbf{R}$ and an embedded cover $F$. The queries are represented as tableaux and are evaluated against the representative instance. He derived an algorithm that translates such a tableau $T$ into a union of tableaux that has the same value of $T_r$ but can be applied to the database relations. As a special case, the algorithm generates efficiently, for any $X \subseteq U$, a union of tableaux that has the same value as the $X$-total projection of the representation instance. On the other hand, the other authors have not proposed expressions like scjes as a means to simulate the total projections nor have they studied formally the optimization process for the expressions generated.

Our strategy allows an efficient implementation: essentially, a number of precomputations (including the closures of all the sets of attributes involved in fds, and some subexpressions associated with the attributes) are performed only once in the beginning, when the scheme is defined (and its independence verified); then, each time a total projection is needed, the union of scjes is generated efficiently. Finally, the optimization techniques studied in [3] can be used, and the optimal expression can be produced very efficiently.

## Acknowledgment

The authors would like to thank F.H. Lochovsky and A.O. Mendelzon for their helpful comments on an initial version of this paper. This research was supported in part by Consiglio Nazionale delle Ricerche, Italy, and by the Natural Sciences and Engineering Research Council of Canada.

## References

[1] A.V. Aho, C. Beeri and J.D. Ullman, The theory of joins in relational databases, *ACM Trans. Database Systems* 4(3) (1979) 297-314.

[2] A.V. Aho, Y. Sagiv and J.D. Ullman, Equivalence of relational expressions, *SIAM J. Comput.* **8**(2) (1979) 218-246.
[3] P. Atzeni and E.P.F. Chan, Efficient optimization of simple chase join expressions, *ACM Trans. Database Systems* **14**(2) (1989) 212-230.
[4] P. Atzeni and D.S. Parker, Assumptions in relational database theory, in: *Proc. ACM PODS* (1982) 1-9.
[5] C. Beeri, A.O. Mendelzon, Y. Sagiv and J.D. Ullman, Equivalence of relational database schemes, *SIAM J. Comput.* **10**(2) (1981) 352-370.
[6] E.P.F. Chan, Query answering and schema analysis under the weak instance model, Ph.D. Thesis, CSRI-164, University of Toronto, 1984.
[7] E.P.F. Chan and A.O. Mendelzon, Independent and separable database schemes, *SIAM J. Comput.* **16**(5) (1987) 841-851.
[8] E.F. Codd, A relational model for large shared data banks, *Comm. ACM* **13**(6) (1970) 377-387.
[9] M.H. Graham and A.O. Mendelzon, On total projections computable by relational algebra, unpublished manuscript, 1982.
[10] M.H. Graham, A.O. Mendelzon and M.Y. Vardi, Notions of dependency satisfaction, *J. ACM* **33**(1) (1986) 105-129.
[11] M.H. Graham and M. Yannakakis, Independent database schemas, *J. Comput. Systems Sci.* **28** (1984) 121-141.
[12] P. Honeyman, Extension joins, in: *Proc. VLDB* (1980) 239-244.
[13] P. Honeyman, Testing satisfaction of functional dependencies, *J. ACM* **29**(3) (1982) 668-677.
[14] M. Ito, M. Iwasaki and T. Kasami, Some results on the representative instance in relational databases, *SIAM J. Comput.* **14**(2) (1985) 334-354.
[15] W. Kent, Consequencies of assuming a universal relation, *ACM Trans. Database Systems* **6**(4) (1982) 539-556.
[16] D. Maier, A.O. Mendelzon and Y. Sagiv, Testing implications of data dependencies, *ACM Trans. Database Systems* **4**(4) (1979) 455-469.
[17] D. Maier, D. Rozenshtein and D.S. Warren, Windows functions, in: *Advances in Computing Research*, Vol. 3 (JAI Press, 1986) 213-246.
[18] D. Maier, J.D. Ullman and M.Y. Vardi, On the foundations of the universal relation model, *ACM Trans. Database Systems* **9**(2) (1984) 283-308.
[19] Y. Sagiv, Can we use the universal instance assumption without using nulls? in: *Proc. ACM SIGMOD* (1981) 108-120.
[20] Y. Sagiv, A characterization of globally consistent database and their correct access paths, *ACM Trans. Database Systems* **8**(2) (1983) 266-286.
[21] Y. Sagiv, Evaluation of queries in independent database schemes, *J. ACM*, to appear.
[22] Y. Sagiv and M. Yannakakis, Equivalence among relational expressions with the union and difference operators. *J. ACM* **27**(4) (1980) 633-655.
[23] J.D. Ullman, The U.R. strikes back, in: *Proc. ACM PODS* (1982) 10-22.
[24] J.D. Ullman, Universal relation interfaces for database systems, in: *Proc. IFIP* (1983) 243-252.
[25] Y. Vassiliou, A formal treatment of imperfect information in data management, CSRG TR-123, University of Toronto, November 1980.
[26] M. Yannakakis, Algorithms for acyclic database schemes, in: *Proc. VLDB* (1981) 82-94.