Optimal Computation of Total Projections with Unions of Simple Chase Join Expressions

Edward P.F. Chan

Computer Systems Research Group
University of Toronto
Toronto, Canada M5S 1A4

ABSTRACT

The representative instance has been proposed as a query answering device in systems using the Universal Relation Interface. One approach is to use the total projections of the representative instance to generate the answer for a query. Associated with this approach is the problem of how to generate the total projections of the representative instance efficiently. We propose a generalization of extension joins, called chase join expressions, as a means to compute the total projections when functional dependencies are given as constraints. In particular, we identify an important subclass of chase join expressions called simple chase join expressions and show that the total projections with respect to a set of functional dependencies can be computed by unions of simple chase join expressions when an independent scheme is assumed. We also find a simple and efficient algorithm that minimizes the number of join operations in a union of simple chase join expressions.

1. Introduction

In recent years, the Universal Relation Interface has been proposed to enhance user performance by relieving users from specifying logical connections among relations in their queries [CK][KU][MUV][O][S1][SP]. A fundamental problem facing systems using the Universal Relation Interface is the problem of defining and computing the connection among a set of attributes $X \subseteq U$. Schenk and Pinkert [SP] implicitly assumed all join sequences containing $X \subseteq U$ return the same result and an algorithm to find such a join sequence was given. In fact the join sequences produced are extension joins [H1]. With a 3NF, cover embedding and lossless join decomposition database scheme, Osborn [O] derived an algorithm which generates a union of extension joins to compute the connection among a set of attributes X, for any $X \subseteq U$. In the experimental query system "q" in Bell Laboratories, a list of stored relations and procedures for computing virtual relations must be specified. The connection among a set of attributes X is found by searching the list for the first relation R such that $X \subseteq R$ and letting $\pi_X(r)$ be the connection among X [AK]. In System/U, it is assumed that all relations are joined completely and the connection among X is defined by $\pi_X(\bowtie_{i=1}^{k} r_i)$ [KU][U]. The expression is then minimized under the weak equivalence. However, this heuristic does not seem to work when the database scheme is cyclic. In this case, maximal

objects are used to break up cycles and limit the range of connections [MU]. It was argued that dependency constraints are not sufficient to capture all the semantic requirements of an application. A data model was proposed to allow a database designer to specify explicitly connections among sets of attributes [MW]. The model is known as the Association-Object Data Model. In this model, associations are the basic units of updates and storage. However not all meaningful connections are contained within associations and objects are used to specify meaningful connections that cannot be captured by associations. The concept of objects is essentially a generalization of maximal objects in [MU]. The connection among X in this model is defined by the union of projections on X of all objects containing X [MW]. Yet another approach is based on the total projections of the representative instance, as was first proposed in [S1][S2]. In this approach, the representative instance is believed to be a correct representation of the information content of a database. For any tuple in the representative instance, the nonnull portion is assumed to be related in a significant way. Hence, the nonnull portion or the X-total projection is assumed to represent the connection among a set of attributes X.

Example 1: $\mathbf{R} = \{R_1(Course, Tutor, Instructor, Dept), R_2(Course, Tutor, Room, Dept)\}$. $F = \{Course \rightarrow Instructor, Course \rightarrow Dept\}$.

Consider the following database state: $r_1=\{<c_1, t_1, i_1, d_1>\}$, $r_2=\{<c_1, t_2, r_1, d_1>\}$. Suppose we want to know who are the instructor and tutors of a course. That is, we want to know the *Course_Instructor_Tutor* relationship. In the total projection approach, we first set up the tableau for the database state T_r as follows:

Course	*Tutor*	*Instructor*	*Dept*	*Room*
c_1	t_1	i_1	d_1	
c_1	t_2		d_1	r_1

We then perform a chase [MMS] on it to obtain the representative instance. From the functional dependency *Course* → *Instructor*, every course has a unique instructor. Since both tuples in T_r have the *Course*-component equal, we can infer that t_2 is also working for instructor i_1. Since no more inference can be made from the state and the constraint, the representative instance of the state is as follows:

Course	*Tutor*	*Instructor*	*Dept*	*Room*
c_1	t_1	i_1	d_1	
c_1	t_2	i_1	d_1	r_1

Hence the *Course_Instructor_Tutor* relationship is defined by $\{<c_1, i_1, t_1>, <c_1, i_1, t_2>\}$. □

In this approach, the most straightforward way of finding the X-total projections is to generate the representative instance and perform total projections on it. However, this takes time and space polynomial proportional to the size of the database even if functional dependencies are given and this may 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 [Y1]. Sagiv considered certain classes of database schemes and showed that the X-total projections can be computed using unions of extension joins [S1][S2].

We believe Sagiv's approach is an interesting and attractive one, since generating the representative instance may be too costly. Furthermore, existing systems do not provide facilities to support the chase procedure. In this paper, we generalize the concept of extension joins [H1] and propose a class of expressions called chase join expressions[†] as a means 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

[†]Independently, a similar class of expressions called *FD-joins* was proposed in [MRW].

simulate the representative instance without physically constructing the tableau. For instance in Example 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 $Course \rightarrow Instructor$, we know each course has a unique instructor associated with it. Therefore by a simple analysis, the *Course_Instructor_Tutor*-total projection can be computed by the following expression:

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

In fact the expression is a union of an important subclass of chase join expressions called simple chase join expressions. To illustrate that chase join expressions are a natural way to simulate the representative instance, we show that for any $X \subseteq U$, the X-total projection with respect to a set of functional dependencies for an independent scheme can be computed by a union of simple chase join expressions. This generalizes the results in [S1][S2]. Since the join operation is the most expensive operation in a relational system and since efficiency is our prime concern, a simple and efficient algorithm is derived to minimize[‡] the number of join operations in a union of simple chase join expressions.

Having defined the necessary notation in Section 2, in Section 3, we derive an algorithm that generates unions of simple chase join expressions to compute the X-total projection with respect to a set of functional dependencies when an independent scheme is given. In order to optimize a union of expressions, we need a characterization of containment and equivalence of expressions. The equivalence problem has been studied in [GM1] and in Section 4, we extend their results and characterize when one expression contains the other over the consistent states. Having known how to test containment and equivalence of expressions, we show how to optimize a union of simple chase join expressions efficiently in Section 5. Finally in Section 6, we conclude with a discussion of some remaining problems.

[‡]Minimization (or optimization) of an expression means finding an equivalent expression with the minimal number of join operations.

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 *tuple* defined on $R = \{A_1, \ldots, A_j\}$ is a function μ that maps each A_i to a value, $1 \le i \le j$. The value can either be a blank, a constant or a variable taken from an infinite set of uninterpreted symbols. If μ is a tuple on R and X is a subset of R, $\mu[X]$ denotes the *restriction* of μ to X. We say $\mu[X]$ is *total* if $\mu[A_i]$ is a constant, for all $A_i \in X$. Let $\pi_X^{\downarrow}$ be the *restricted* or *total projection operator* which is defined as $\pi_X^{\downarrow}(I) = \{t[X] \mid t \in I$ and $t[X]$ is total$\}$. Let π denote the usual projection operator.

2.2 Consistency, Expression Values, Equivalence and Containment of Expressions

Let $\mathbf{R} = \{R_1, \ldots, R_k\}$ be a database scheme. A database state r on $\mathbf{R}$ is said to be *consistent* with a set of dependencies Σ if a weak instance exists for the state with respect to Σ [GMV][H2][V]. Let $WSAT(\mathbf{R}, \Sigma) = \{r \mid r$ is defined on $\mathbf{R}$ and r is consistent with $\Sigma\}$.

We shall consider relational expressions in which the only operators are select, project, (natural) join and union. If only the operators select, project and join are involved, the expressions are called *SPJ-expressions*. The operands are relation schemes in a database scheme. 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 the operators applied 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 $E_2 \supseteq E_1$.

2.3 Tableaux and Containment Mappings

A *tableau* is a relation defined on U. Each column of a tableau corresponds to an attribute in U. The domain of the i^{th} column of the tableau, corresponding to some attribute A_i, consists of the *distinguished variable (dv)* a_i, countable many *nondistinguished variables (ndv's)* $\{b_{ij}\}$ and *constants* taken from the domain of A_i.

The first row of a tableau (if there is any) is called the *summary* and only contains dv's, constants and blanks. The attributes corresponding to nonblank columns of the summary define the *target relation scheme* of the tableau. All the other rows in a tableau are simply called *rows*. A row of a tableau may contain dv's, ndv's and constants. No variables can appear in two different columns in a tableau, and a dv does not appear in a column unless it also appears in the summary of that column [ASU1]. Each row in a tableau is optionally tagged with some relation scheme R_i.

A *valuation function* $\nu : S_1 \rightarrow S_2$ is a functional mapping on symbols in set S_1 to symbols in set S_2 with the following restrictions:

- ν maps a ndv to a ndv, a dv or a constant.
- ν maps a dv to a dv or a constant.
- ν is an identity mapping on constants.

If ν is also an identity mapping on tags then ν is called a *tag preserving valuation function*. Unless otherwise stated, all valuation functions are tag preserving. A *containment mapping* ν from 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 if t_i is a row in T_1 then $\nu(t_i)$ is a row in T_2. A tableau T_1 is said to *contain* T_2, written $T_1 \supseteq T_2$, if they define the same target relation scheme and there is a containment mapping from T_1 to T_2 [ASU1]. T_1 is said to be *equivalent* to T_2, denoted by $T_1 \equiv T_2$, if $T_1 \supseteq T_2$ and $T_2 \supseteq T_1$.

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

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

The summary of T_r is undefined. Let E be a SPJ-expression. Then we can always construct a tagged tableau T_E for E. Furthermore, $E(r) = T_E(T_r)$, for any state r [ASU1].

2.4 Functional Dependencies, Transformations and Chasing

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

A fd $X \rightarrow A$ is said to be *embedded* in a relation scheme R if $XA \subseteq R$. The projection of a set of fd's F onto R_i, denoted by $F^+ | R_i$, is the set of *projected fd's* $X \rightarrow A \in F^+$ such that XA is embedded in R_i. A database scheme $\mathbf{R}$ is said to be *cover embedding* for a set of fd's F if there exists a set of fd's G with $G^+ = F^+$ such that for each fd $X \rightarrow A \in G$, $X \rightarrow A$ 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 [BMSU], $\mathbf{R}$ is cover embedding implies $\mathbf{R}$ is dependency preserving.

Let F be a set of fd's. We use $\tau=<l_s, l_t, X \rightarrow A>$ to represent a *transformation* applying to a tableau T, where l_s and l_t are rows in T and $X \rightarrow A \in F^+$. A transformation τ is *valid* if $l_s[X] = l_t[X]$. A transformation τ is *applied* to a tableau T, denoted by $\tau(T)$, if τ is valid and T is changed according to the following ways:

- If both $l_s[A]$ and $l_t[A]$ are distinct ndv's then replace the one with lower subscript with the higher one. If one of them is a ndv but the other is not, replace the ndv with the other. If one is a dv and the other is a constant, replace the dv with the constant. If both are distinct constants then $\tau(T) = \emptyset$ and T is said to *contradict* τ.

Let $\tau=\tau_1 \cdots \tau_p$ be a sequence of transformations. Then $\tau(T) = \tau_p(\tau_{p-1}(\cdots \tau_1(T) \cdots))$. Let Σ 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 Σ exhaustively to T_r. Given a representative instance $CHASE_\Sigma(T_r)$, the *X-total projection* on $CHASE_\Sigma(T_r)$ is defined as $\pi_X^\downarrow(CHASE_\Sigma(T_r))$. 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 finite predetermined relational expression [MUV][GM2].

2.5 Derivation Sequences and Chase Join Expressions

Given a set of fd's F, a *derivation sequence (ds)* of some relation scheme R_{i_0} is a finite sequence of fd's $\{f_1 : Y_1 \rightarrow Z_1, \ldots, f_m : Y_m \rightarrow Z_m\}$ satisfying the following conditions:

- $Y_j \rightarrow Z_j \in F^+$, for all $1 \le j \le m$.
- $Y_j \subseteq (\bigcup_{k=1}^{j-1} Y_k Z_k \cup R_{i_0})$ and $Z_j \cap (\bigcup_{k=1}^{j-1} Y_k Z_k \cup R_{i_0}) = \emptyset$, for all $1 \le j \le m$.

The ds is said to have a *length* of m. Essentially a ds of R_{i_0} is a sequence of fd's used in computing the closure of R_{i_0}. A ds of R_{i_0} *covering* X is a ds of R_{i_0} such that $(\bigcup_{j=1}^{m} Y_j Z_j \cup R_{i_0}) \supseteq X$.

A ds is said to be *decomposed* if for every fd $Y_i \rightarrow Z_i$ 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 $Y_i \rightarrow Z_i$ with $Y_i \rightarrow A_{j_1}, \ldots, Y_i \rightarrow A_{j_p}$, where $Z_i = A_{j_1} \cdots A_{j_p}$, for all $Y_i \rightarrow Z_i$ in the ds. Two decomposed ds's of R_{i_0} are said to be *equivalent* if they are identical up to permutation of fd's in the sequences. Two ds's are *equivalent* exactly when their decomposed forms are. A decomposed ds of R_{i_0} covering X is said to be *nonredundant* if for all $1 \le j \le m$, $f_j : Y_j \rightarrow A_j$, either $A_j \in X$ or $A_j \in Y_k$, for some $k > j$. A ds of R_{i_0} covering X is said to be *nonredundant* if its decomposed form is.

Example 2: $\mathbf{R} = \{R_1(AB), R_2(AC), R_3(BD)\}$. $F = \{A \rightarrow C, B \rightarrow D\}$.

The following are two ds's of R_1 covering AD: $\{A \rightarrow C, B \rightarrow D\}$, $\{B \rightarrow D, A \rightarrow C\}$. The two ds's are equivalent. Consider the ds $\{A \rightarrow C, B \rightarrow D\}$. $A \rightarrow C$ is redundant for $C \notin AD$ and $C \neq B$. Hence $\{B \rightarrow D\}$ is a nonredundant ds of R_1 covering AD. □

Let R_1 and R_2 be a pair of relation schemes for which there exists $Y \subseteq R_2 - R_1$ such that $R_1 \cap R_2 \rightarrow Y \in F^+$. The join of r_1 and $\pi_{(R_1 \cap R_2) \cup Y}(r_2)$ is called an *extension join* [H1]. A generalization of extension joins is called *chase join expressions (cje's)* and are recursively defined as follows:

(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 cje's defined on R and S respectively. Then $E_i \bowtie \pi_{XY}(E_j)$ is a cje defined on $R \bigcup Y$, where $X \subseteq R \bigcap S$, $Y \subseteq S - R$, and $X \rightarrow Y \in F^+$.

Given a ds $\{Y_1 \rightarrow Z_1, \ldots, Y_m \rightarrow Z_m\}$ 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:

$$E = \pi_X(R_{i_0} \bowtie \pi_{Y_1Z_1}(E_{i_1}) \bowtie \cdots \bowtie \pi_{Y_mZ_m}(E_{i_m})),$$

where for each $1 \leq j \leq m$, Y_jZ_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*. Otherwise it is a *complex* cje. In the subsequent discussion, we use cje to mean cje for some ds of a relation scheme covering X, for some $X \subseteq U$.

The following example demonstrates how cje's can be used to compute the X-total projections when a set of fd's F is given as the constraint.

Example 3: $\mathbf{R} = \{R_1(Course, Teacher), R_2(Course, Hour, Room), R_3(Teacher, Hour, Phone)\}$, $F = \{Course \rightarrow Teacher, Teacher\ Hour \rightarrow Room\}$.

R_1 represents the *Course_Teacher* relationship. R_2 keeps track of the time and place a course is held. The phone number a teacher can be reached at a specific time is stored in R_3. The constraints that each course is taught by a unique teacher and that a teacher can be in only one place at a time are represented by $Course \rightarrow Teacher$ and $Teacher\ Hour \rightarrow Room$ respectively. Let T_r be a tableau for a consistent state r. $CHASE_F(T_r)$ is obtained from T_r by applying transformations to T_r.

Suppose t_1 is a tuple in $CHASE_F(T_r)$ that originates from r_1. Since $t_1[Hour\ Phone]$ are assigned with distinct null symbols in T_r and there is no fd of the form $Y \rightarrow Hour$ or of the form $Y \rightarrow Phone$, $t_1[Hour\ Phone]$ are distinct nulls in $CHASE_F(T_r)$. Since $t_1[Hour]$ is a distinct null symbol, any transformation involving t_1 will not involve the fd $Teacher\ Hour \rightarrow Room$. Therefore $t_1[Room]$ is also a distinct null symbol in $CHASE_F(T_r)$. Hence the *Course_Teacher*-components of t_1 are the only components containing constants in $CHASE_F(T_r)$ and all other components are distinct nulls.

Consider a tuple t_2 in $CHASE_F(T_r)$ that originates from r_2. Clearly *Course_Hour_Room*-components are constants. The *Teacher*-component of t_2 is constant if and only if there exists a tuple t_1 that originates from r_1 with $t_1[Course] = t_2[Course]$. $t_2[Phone]$ cannot be constant for there is no fd of the form $Y \rightarrow Phone$ in F. Therefore for any tuple t_2 that originates from r_2 in $CHASE_F(T_r)$, the nonnull portion can be obtained by applying at most one transformation to T_r. In particular, the transformation is of the form $< t_1, t_2, Course \rightarrow Teacher >$, where t_1 is a tuple that originates from r_1.

Consider a tuple t_3 that originates from r_3 in $CHASE_F(T_r)$. Clearly the *Teacher_Hour_Phone*-components are constants in $CHASE_F(T_r)$. The *Course*-component can never be constant for there is no fd of the form $Y \rightarrow Course$ in F. The *Room*-component of t_3 is constant if and only if there exists a t_2 that originates from r_2 in $CHASE_F(T_r)$ such that $t_2[Teacher\ Hour] = t_3[Teacher\ Hour]$. Since $t_2[Teacher]$ could become constant after application of one transformation, $t_3[Room]$ can be derived by applying two transformations to T_r. Therefore any constant components of any tuple in $CHASE_F(T_r)$ can be obtained by applying at most two transformations. Hence $\mathbf{R}$ is bounded.

Suppose we want to compute the *Teacher_Hour_Room_Phone*-total projection. That is we want to find the teacher, time, place and phone relationship. Any tuple in $CHASE_F(T_r)$ whose *Teacher_Hour_Room_Phone*-components are constants are those tuples that originate from r_3. From the analysis presented above, the following complex cje returns the *Teacher_Hour_Room_Phone*-total projection for those tuples that originate from r_3:

$\pi_{Teacher, Hour, Room, Phone}(R_3 \bowtie \pi_{Teacher, Hour, Room}(R_2 \bowtie \pi_{Course, Teacher}(R_1)))$.

The subexpression $\pi_{Teacher, Hour, Room}(R_2 \bowtie \pi_{Course, Teacher}(R_1))$ returns the *Teacher_Hour_Room*-total projection on those tuples that originate from r_2. Hence the subexpression $R_3 \bowtie \pi_{Teacher, Hour, Room}(R_2 \bowtie \pi_{Course, Teacher}(R_1))$ returns the *Teacher_Hour_Room_Phone*-total projection on tuples that originate from r_3. Since tuples in the representative instance whose *Teacher_Hour_Room_Phone*-components are constants must originate from r_3, the above expression is an expression to compute the *Teacher_Hour_Room_Phone*-total projection. □

3. Computing Total Projections for Independent Schemes with Unions of Simple Chase Join Expressions

In this Section, we show that there is an algorithm which generates unions of simple cje's to compute the X-total projections with respect to an embedded cover F when an independent scheme is considered. Before we give the algorithm, some fundamental properties of independent schemes are needed.

Independent schemes were defined in [GY] and a polynomial algorithm was given to recognize an independent scheme when $\Sigma = H \bigcup \{\bowtie \mathbf{R}\}$ is given as the constraint, where H is a set of fd's. A database scheme $\mathbf{R}$ is independent 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 fd's.

Theorem 3.1: The following are equivalent.

(1) $\mathbf{R}$ is independent with respect to Σ.

(2) $\mathbf{R}$ is independent with respect to F, where F is the set of fd's implied by Σ.

(3) If for each $r_i \in r$, r_i satisfies $F^+|R_i$, then r is a consistent state for any r defined on $\mathbf{R}$.

[Proof]: See [GY]. □

In fact, $\mathbf{R}$ is cover embedding with respect to F. In [GY] an algorithm was derived to find an embedded cover F if $\mathbf{R}$ is independent with respect to Σ. In the subsequent discussion we assume the embedded cover is given 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 $F^+|R_i$, for any $R_i \in \mathbf{R}$. Also, all ds's are assumed to be decomposed. The following algorithm will convert T_r to $CHASE_F(T_r)$.

Algorithm 1: Chasing the tableau for an independent state.

(1) Finish = false. Loop until finish = true: If there is an $X \rightarrow A \in F$ and t_1, t_2 in T_r such that $t_1[XA]$ are constants, $t_1[X] = t_2[X]$ 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 $CHASE_F(T_r)$. □

During step (2) of Algorithm 1, only null symbols are equated.

Lemma 3.1: Let $\mathbf{R}$ be an independent scheme with respect to an embedded cover F, where F is a set of fd's. Let $r \in WSAT(\mathbf{R}, F)$. Consider the chasing of T_r using Algorithm 1, then in step (2) of the Algorithm only null symbols are equated.

[Proof]: See [C]. □

The following Lemma shows that each derived value in $CHASE_F(T_r)$ is "uniquely" derived from a relation for an independent scheme.

Lemma 3.2: Let $\mathbf{R}$ be an independent scheme with respect to an embedded cover F, where F is a set of fd's. Let $r \in WSAT(\mathbf{R}, F)$ and for any nonredundant cover F_j of $F^+|R_j$, for any $R_j \in \mathbf{R}$ and for any $X \rightarrow A \in F_j$, $\pi^{\downarrow}_{XA}(CHASE_F(T_r)) = \pi_{XA}(r_j)$.

[Proof]: See [CM]. □

Let $r \in WSAT(\mathbf{R}, F)$ and let $t \in CHASE_F(T_r)$, where t originates from r_i. Define $T = \{A \mid t[A]$ becomes constant in the chase process$\} = R_iA_1 \cdots A_k$, $k \geq 0$. Using Algorithm 1 and by Lemma 3.1, there exists a sequence of transformations χ that changes $t[A_1 \cdots A_k]$ from nulls to constants. Without loss of generality, we assume $t[A_1 \cdots A_k]$ become constants in the order $A_1 \cdots A_k$. Clearly each entry $t[A_i]$ is changed by exactly one transformation. Therefore $|\chi|=k$ and let the fd's involved in χ be $\{X_1 \rightarrow A_1, \ldots, X_k \rightarrow A_k\} = \{f_1, \ldots, f_k\}$, $k \geq 0$. 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_j is embedded in exactly one relation scheme, let the relation scheme be R_{i_j}.

It is clear that $\{f_1, \ldots, f_k\}$ is a ds of R_i covering $\{A_1, \ldots, A_k\}$. This follows directly from step (1) of Algorithm 1. We claim that the following simple cje E for the ds $\{f_1, \ldots, f_k\}$ produces a relation containing $t[T]$,

$E = R_i \bowtie \pi_{X_1A_1}(R_{i_1}) \bowtie \cdots \bowtie \pi_{X_kA_k}(R_{i_k})$, where R_{i_j} embeds f_j.

Lemma 3.3: Let t, T and E be defined above. $t[T] \in E(r)$.

[*Proof*]: Prove by induction on the number 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 $t[R_iA_1 \cdots A_{k-1}]$ are constants after t is changed by $k-1$ transformations in step (1) of Algorithm 1. Suppose $t[R_iA_1 \cdots A_k]$ become constants in the chase process. Let $\chi = \langle s, t, X_k \rightarrow A_k \rangle$ be the transformation that transforms $t[A_k]$ from null to constant in step (1) of Algorithm 1, where $X_k \rightarrow A_k$ is embedded in R_{i_k}. By the induction hypothesis, $t[R_iA_1 \cdots A_{k-1}]$ is contained in some simple cje E for a ds of R_i, where $E = R_i \bowtie \pi_{X_1A_1}(R_{i_1}) \bowtie \cdots \bowtie \pi_{X_{k-1}A_{k-1}}(R_{i_{k-1}})$. We want to show that $t[R_iA_1 \cdots A_k]$ is contained in the simple cje $E \bowtie \pi_{X_kA_k}(R_{i_k})$.

Before χ is applied $s[X_kA_k]$ are constants. By the assumption of F and by Lemma 3.2, $s[X_kA_k] \in \pi_{X_kA_k}(r_{i_k})$. Hence $t[R_iA_1 \cdots A_k] \in E \bowtie \pi_{X_kA_k}(r_{i_k})$. This completes the induction proof. □

Next, we want to show that in general given any simple cje for a ds of R_i, it only produces total tuples in $CHASE_F(T_r)$. Let a ds of R_i be $\{f_1 : X_1 \rightarrow A_1, \ldots, f_k : X_k \rightarrow A_k\}$. Let $E = R_i \bowtie \pi_{X_1A_1}(R_{i_1}) \bowtie \cdots \bowtie \pi_{X_kA_k}(R_{i_k})$ be a simple cje.

Lemma 3.4: Let $r \in WSAT(\mathbf{R}, F)$, $\mathbf{R}$ is any database scheme. Let E be defined above. For any $t \in E(r)$, $t = s[T]$, for some s in $CHASE_F(T_r)$.

[*Proof*]: This can easily be proven by a simple induction on the number of joins in E. □

By Lemmas 3.3 and 3.4, the set of all simple cje's for ds's of R_i's produces exactly the set of total tuples in the representative instance. Hence we have an algorithm to compute the X-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 $R_i \in \mathbf{R}$. Then for each $R_i \in \mathbf{R}$ such that $R_i^+ \supseteq X$, find all nonequivalent ds's of R_i covering X. The ds's are of the form $\{X_1 \rightarrow A_1, \ldots, X_m \rightarrow A_m\}$, where $X_j \rightarrow A_j \in F$, for all $1 \leq j \leq m$. For each of these ds's, construct a simple cje for it. It is worth mentioning that the simple cje constructed is in fact unique. The union of all these simple cje's is an expression for computing the X-total projection.

Theorem 3.2: 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 $E = \bigcup_i \pi_X(E_i)$. That is E is the union of all simple cje's obtained as described above.

[*Proof*]: Follows from Lemmas 3.3 and 3.4. □

In computing the X-total projections, we do not have to enumerate all ds's of R_i covering X, $R_i \in \mathbf{R}$. In fact we will show that we only need to consider the nonredundant ones. This follows from a more general result.

Suppose $R_i^+ \supseteq X$ and let $\tau=\{f_1 : Y_1 \rightarrow A_1, \ldots, f_m : Y_m \rightarrow A_m\}$ be a redundant ds for R_i covering X. Let $\chi=\{f_1' : Y_1' \rightarrow A_1', \ldots, f_n' : Y_n' \rightarrow A_n'\}$ be a nonredundant ds of τ. Let the redundant fd's removed from τ be $\tau' = \{g_1 : Z_1 \rightarrow B_1, \ldots, g_t : Z_t \rightarrow B_t\}$. Let E_τ be a cje for τ. Let E_χ be a nonredundant cje for χ and is obtained from E_τ by deleting those subexpressions corresponding to redundant fd's in τ'.

Lemma 3.5: Let E_τ and E_χ be a redundant and its corresponding nonredundant cje respectively. Then $E_\chi \supseteq E_\tau$.

[*Proof*]: This can easily be proven by using the fact that joins are associative. □

Example 4: $\mathbf{R} = \{R_1(AB), R_2(ABCDEF)\}$. $F = \{AB \rightarrow D, BC \rightarrow E, B \rightarrow C, D \rightarrow F, E \rightarrow F\}$.

$\mathbf{R}$ is an independent scheme and $F = F_2$ is a nonredundant cover for $F^+|R_2$. Suppose we want to find the $ABCF$-total projection. Since $R_1^+ \supseteq ABCF$ and $R_2^+ \supseteq ABCF$, we consider them individually. With F, the following are all the nonequivalent and nonredundant ds's of R_1 covering $ABCF$.

(1) $B \rightarrow C, BC \rightarrow E, E \rightarrow F$.

(2) $AB \rightarrow D, D \rightarrow F, B \rightarrow C$.

The simple cje's for them are

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

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

Since $R_2 \supseteq ABCF$, the only nonredundant ds is the empty sequence and hence the simple cje for R_2 is

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

Therefore the $ABCF$-total projection is computed by $E_1 \bigcup E_2 \bigcup E_3$. □

As a consequence of Theorem 3.2, independent schemes are bounded with respect to the embedded cover F.[†]

Corollary 3.1: Let $\mathbf{R}$ be an independent scheme with respect to an embedded cover F. Then $\mathbf{R}$ is bounded with respect to F.

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 $\Sigma = F \bigcup \{\bowtie\mathbf{R}\}$ is given as a constraint. It is proven in [CM] that given a dependency preserving scheme $\mathbf{R}$, for any $R_i \in \mathbf{R}$, and for any $X \subseteq R_i$, $\pi_X^\downarrow(CHASE_F(T_\tau)) = \pi_X^\downarrow(CHASE_\Sigma(T_\tau))$. Since independent schemes are cover embedding and hence dependency preserving, the X-total projection with respect to Σ 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 [ABU], it is easy to verify that $\pi_X^\downarrow(CHASE_F(T_\tau)) = \pi_X^\downarrow(CHASE_\Sigma(T_\tau))$, for any $X \subseteq U$. Hence our method is still applicable in these cases.

4. A Characterization of Equivalence and Containment of Expressions

In [GM1], a characterization of when two expressions are equivalent over the consistent states was derived. We extend their work a bit here and characterize when one expression is contained by the other. As in [GM1], the set of constraints Σ considered is a set of *equality generating dependencies* and *total tuple generating dependencies* [BV][F]. Also all expressions are assumed to be defined on the same set of attributes.

Theorem 4.1: Let E_1, E_2 be two SPJ-expressions. $E_1 \equiv E_2$ if and only if $CHASE_\Sigma(T_{E_1}) \equiv CHASE_\Sigma(T_{E_2})$.

[†] Independently Maier et al [MRW] obtained a similar result.

[*Proof*]: See [GM1]. □

Lemma 4.1: Let T be any tableau. Then there exists a containment mapping from T to $CHASE_\Sigma(T)$.

[*Proof*]: This can be proven by a simple induction on the number of transformations applied to T. □

Lemma 4.2: Let T, T' be two tableaux and ν a containment mapping from T to T'. χ is a valid sequence of transformations for T. The transformations χ_ν are formed by composing χ with the mapping in ν. Then χ_ν is valid for T' and ν is a containment mapping from $\chi(T)$ to $\chi_\nu(T')$, provided $\chi_\nu(T') \neq \emptyset$.

[*Proof*]: See [GM1]. □

Lemma 4.3: Let r be a consistent state. For any SPJ-expression E, $T_E(T_r) = CHASE_\Sigma(T_E)(CHASE_\Sigma(T_r))$.

[*Proof*]: See [GM1]. □

Lemma 4.4: Let E_1, E_2 be two SPJ-expressions. $E_1 \supseteq E_2$ implies $T_{E_1} \supseteq CHASE_\Sigma(T_{E_2})$.

[*Proof*]: See [GM1]. □

Theorem 4.2: Let E_1, E_2 be two SPJ-expressions. The following are equivalent.

(1) $E_1 \supseteq E_2$.

(2) $CHASE_\Sigma(T_{E_1}) \supseteq CHASE_\Sigma(T_{E_2})$.

(3) $T_{E_1} \supseteq CHASE_\Sigma(T_{E_2})$.

[*Proof*]: (1)⇔(2). (If) Given any consistent state r, we want to show that $E_1 \supseteq E_2$. By Lemma 4.3 $T_{E_1}(T_r) = CHASE_\Sigma(T_{E_1})(CHASE_\Sigma(T_r)) \supseteq CHASE_\Sigma(T_{E_2})(CHASE_\Sigma(T_r)) = T_{E_2}(T_r)$. Hence $E_1 \supseteq E_2$.

(Only if) By Lemma 4.4, $T_{E_1} \supseteq CHASE_\Sigma(T_{E_2})$. Hence there exists a containment mapping ν from T_{E_1} to $CHASE_\Sigma(T_{E_2})$. Apply a sequence of transformations χ to T_{E_1} to obtain $CHASE_\Sigma(T_{E_1})$. Let χ_ν be the sequence of transformations obtained by composing χ with the mapping ν. By Lemma 4.2, there exists a containment mapping ν from $CHASE_\Sigma(T_{E_1})$ to $CHASE_\Sigma(T_{E_2})$.

(2)⇔(3). (If) Using an argument similar to (Only if) above, there exists a containment mapping from $CHASE_\Sigma(T_{E_1})$ to $CHASE_\Sigma(T_{E_2})$. Hence $CHASE_\Sigma(T_{E_1}) \supseteq CHASE_\Sigma(T_{E_2})$.

(Only if) There exists a containment mapping from $CHASE_\Sigma(T_{E_1})$ to $CHASE_\Sigma(T_{E_2})$. By Lemma 4.1, there exists a containment mapping from T_{E_1} to $CHASE_\Sigma(T_{E_1})$. Since composition of containment mappings is still a containment mapping, $T_{E_1} \supseteq CHASE_\Sigma(T_{E_2})$. □

5. Efficient Optimization of Unions of Simple cje's

Before we show how to test containment and how to minimize simple cje's, we need to know some properties of this class of expressions. Let us consider a more general class of expressions. Let $\tau = \{f_1 : Y_1 \rightarrow Z_1, \ldots, f_m : Y_m \rightarrow Z_m\}$ be a ds of R_{i_0}. Let $E = \pi_X(R_{i_0} \bowtie \pi_{W_1}(R_{i_1}) \bowtie \cdots \bowtie \pi_{W_k}(R_{i_k}))$, where $k \leq m$ and for each f_i in τ, there exists at least a W_j such that $W_j \supseteq Y_i Z_i$. Furthermore for each W_j, there exists a nonempty subset $\{Y_{j_1} \rightarrow Z_{j_1}, \ldots, Y_{j_p} \rightarrow Z_{j_p}\}$ of τ such that $W_j = Y_{j_1} Z_{j_1} \cdots Y_{j_p} Z_{j_p}$ and $W_j \subseteq R_{i_j}$. Let T_E be a tagged tableau for E. We use $l_0, \ldots, l_k$ to denote the set of rows in T_E corresponding to the subexpressions $R_{i_0}, \ldots, \pi_{W_k}(R_{i_k})$ respectively. In particular, if there is a one-one correspondence between fd's in τ and W_i's in E, then E is a simple cje for τ.

Given a tableau T_E and $A \in U$, a symbol $l_p[A]$ is said to be *significant* if either $l_p[A]$ is a dv or is a repeated symbol. A symbol is said to be *repeated* if it appears in more than one row. Let l_s and l_t be any two rows in T_E. l_s is said to *contain* l_t, denoted by $l_s \geq l_t$, if for all $A \in U$, $l_t[A]$ is a significant symbol implies $l_s[A]$ is also a significant symbol. The following is a way to obtain $CHASE_F(T_E)$.

Algorithm 2: Given T_E and a set of fd's F, return $CHASE_F(T_E)$.

Input: A set of fd's F and the tableau T_E, where E is an expression defined above.

Output: $CHASE_F(T_E)$.

Method: We chase T_E in the following way:

(1) Apply transformations to T_E such that for all $1 \le j \le m$, $l_0[Z_j]$ is set equal to $l_p[Z_j]$, where W_p embeds Y_jZ_j and no other entry is changed. Let the resulting tableau be T_E'.

(2) Apply transformations to T_E' to obtain $CHASE_F(T_E)$. □

Lemma 5.1: In step (1) of Algorithm 2, we can always get T_E' from T_E by applying m transformations $\chi_1 \cdots \chi_m$ to T_E, where $\chi_j = <l_0, l_{p_j}, Y_j \to Z_j>$, $1 \le j \le m$ and for any W_{p_j} contains Y_jZ_j.

[*Proof*]: This can be proven by a simple induction on the number of fd's in τ. □

By the way T_E is defined, if there are two W_s, W_t embed $f_j: Y_j \to Z_j$, then $l_s[Y_jZ_j]=l_t[Y_jZ_j]$ in T_E. Therefore after step (1), for all W_s's embed Y_jZ_j, $l_s[Y_jZ_j]=l_0[Y_jZ_j]$, for all fd's f_j in τ. Since $W_s=Y_{s_1}Z_{s_1} \cdots Y_{s_p}Z_{s_p}$, after step (1), $l_s[W_s]=l_0[W_s]$, for all $1 \le s \le k$.

Lemma 5.2: Let l_p be a row in $CHASE_F(T_E)$, $p \ge 1$. For all $A \in U$, if $A \notin W_p^+$ then $l_p[A]$ is a distinct ndv in $CHASE_F(T_E)$.

[*Proof*]: Since transformations in step (1) only change entries in l_0, this can be proven by an induction on the number of transformations applied to T_E' in step (2). See [C] for detail. □

The following Theorems and Corollaries summarize some fundamental properties of the chased tableau $CHASE_F(T_E)$.

Theorem 5.1: Let l_p be a row in T_E, $p \ge 1$. Then the following are equivalent.

(1) $A \in W_p^+$.

(2) $l_p[A]=l_0[A]$ in $CHASE_F(T_E)$.

(3) $l_p[A]$ is a significant symbol in $CHASE_F(T_E)$.

[*Proof*]: (1) => (2). After step (1) of Algorithm 1, $l_p[W_p]=l_0[W_p]$, for all $1 \le p \le k$. Hence if $A \in W_p^+$, $l_p[A]=l_0[A]$ in $CHASE_F(T_E)$.

(2) => (3). Follows directly from the definition of significant symbols.

(3) => (1). If $A \notin W_p^+$, by Lemma 5.2, $l_p[A]$ is a distinct ndv in $CHASE_F(T_E)$. □

Corollary 5.1: For each column A in $CHASE_F(T_E)$, there is at most one significant symbol in it.

[*Proof*]: See [C]. □

A tableau is said to be *simple* if in any column with a repeated ndv there is no other repeated symbol in that column [ASU1][ASU2].

Corollary 5.2: $CHASE_F(T_E)$ is a simple tableau.

[*Proof*]: See [C]. □

Theorem 5.2: $l_0[A]$ is a significant symbol in $CHASE_F(T_E)$ if and only if $A \in \bigcup_{p=1}^{k} W_p^+$ or $A \in (X \cap R_{i_0})$.

[*Proof*]: (If) For any attribute $A \in (X \cap R_{i_0})$, $l_0[A]$ is a dv and hence a significant symbol in T_E. Suppose $A \in W_p^+$, for some $1 \le p \le k$. By Theorem 5.1, we know $l_0[A]=l_p[A]$ in $CHASE_F(T_E)$. Therefore $l_0[A]$ is a significant symbol.

(Only if) Suppose there exists $A \notin (X \cap R_{i_0})$ and $A \notin \bigcup_{p=1}^{k} W_p^+$ but $l_0[A]$ is a significant symbol. Since $A \notin (X \cap R_{i_0})$, $l_0[A]$ is a repeated symbol in $CHASE_F(T_E)$. This means there exists l_p, $1 \le p \le k$, such that $l_0[A]=l_p[A]$. By Theorem 5.1, $A \in W_p^+$. A contradiction. □

Corollary 5.3: Let $l_p[A]$ be a repeated symbol in $CHASE_F(T_E)$, for some $0 \le p \le m$. Then there exists l_q in $CHASE_F(T_E)$ such that $p \ne q$, l_p and l_q have different tags and $l_p[A] = l_q[A]$.

[*Proof*]: See [C]. □

We are now ready to show how to optimize a union of simple cje's. Given two simple cje's E_1, E_2 for some ds's of some relation schemes covering X, the following Theorem characterizes when $E_1 \supseteq E_2$ in terms of their chased tableaux when a set of fd's is given.

Theorem 5.3: Let E_1 and E_2 be two simple cje's for some relation schemes covering X. $E_1 \supseteq E_2$ if and only if for each row s_i in $CHASE_F(T_{E_1})$ there exists a row t_j in $CHASE_F(T_{E_2})$ such that s_i and t_j both have the same tag and $t_j \geq s_i$.

[*Proof*]: By Theorem 4.2, $E_1 \supseteq E_2$ if and only if $CHASE_F(T_{E_1}) \supseteq CHASE_F(T_{E_2})$. We want to show that $CHASE_F(T_{E_1}) \supseteq CHASE_F(T_{E_2})$ if and only if the condition in the Theorem is true.

(If) Define a containment mapping ν as follows. For each row s_i in $CHASE_F(T_{E_1})$, there exists t_j in $CHASE_F(T_{E_2})$ such that $t_j \geq s_i$. Then map symbols in s_i to the corresponding symbols in t_j. It is easy to prove that the mapping is a containment mapping.

(Only if) Since $CHASE_F(T_{E_1}) \supseteq CHASE_F(T_{E_2})$, there exists a containment mapping from $CHASE_F(T_{E_1})$ to $CHASE_F(T_{E_2})$. Let s_i be a row in $CHASE_F(T_{E_1})$ and let t_j be the row to which s_i is mapped. Clearly s_i and t_j have the same tag. Let $A \in U$. Consider the following two possible cases.

Case (1): $s_i[A]$ is a dv. In this case $t_j[A]$ must be a dv.

Case (2): $s_i[A]$ is a repeated ndv. By Corollary 5.1, there is no dv in column A, hence $A \notin X$. Since $A \notin X$, there is no dv in column A in $CHASE_F(T_{E_2})$. We claim $t_j[A]$ is also a repeated ndv. Since $s_i[A]$ is a repeated ndv, by Corollary 5.3, there exists another row s_p in $CHASE_F(T_{E_1})$ such that $s_p[A]=s_i[A]$ and they have different tags. If $t_j[A]$ is a distinct ndv then s_p is mapped into t_j. This implies the valuation function is not a tag preserving one. Hence for each $A \in U$, $s_i[A]$ is a significant symbol implies $t_j[A]$ is also a significant symbol. Therefore $t_j \geq s_i$. □

Although the above Theorem characterizes containment of simple cje's in terms of their chased tableaux, it is not necessary to generate the chased tableaux physically at all. By Theorems 5.1 and 5.2, we know exactly which columns in a row of a chased tableau contain significant symbols. Clearly, the testing can be done in $O(m^*n^* |F|)$, where m and n are the space needed to write down E_1 and E_2 respectively. The following Corollary characterizes when one simple cje contains the other if both are simple cje's for some R_{i_0} covering X.

Corollary 5.4: Let $\tau = \{Y_1 \rightarrow Y_1', \ldots, Y_m \rightarrow Y_m'\}$ and $\chi = \{Z_1 \rightarrow Z_1', \ldots, Z_n \rightarrow Z_n'\}$ be two ds's of R_{i_0} covering $X \subseteq U$. Let $E_\tau = \pi_X(R_{i_0} \bowtie \pi_{Y_1Y_1'}(R_{i_1}) \bowtie \cdots \bowtie \pi_{Y_mY_m'}(R_{i_m}))$ and $E_\chi = \pi_X(R_{i_0} \bowtie \pi_{Z_1Z_1'}(S_{i_1}) \bowtie \cdots \bowtie \pi_{Z_nZ_n'}(S_{i_n}))$. $E_\tau \supseteq E_\chi$ if and only if for each $Y_j \rightarrow Y_j'$ in τ there exists a $Z_k \rightarrow Z_k'$ in χ such that $Z_k^+ \supseteq Y_j^+$ and $R_{i_j} = S_{i_k}$.

[*Proof*]: See [C]. □

Example 5: $\mathbf{R} = \{R_1(AB), R_2(ABCDEF)\}$. $F = \{AB \rightarrow D, BC \rightarrow E, B \rightarrow C, D \rightarrow F, E \rightarrow F\}$.

In Example 4, we showed that the $ABCF$-total projection is computed by $E_1 \cup E_2 \cup E_3$ where

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

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

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

Let us consider E_1 and E_2. Since $ABD^+ = ABCDEF = U$, by Corollary 5.4, $E_1 \supseteq E_2$. Since T_{E_1} has a row with tag R_1 and T_{E_3} does not, by Theorem 5.3, E_1 cannot contain E_3. $E_3 \supseteq E_1$ if and only if one of the closures of BC, BCE or EF contains $ABCF$. Since $BC^+ = BCEF$, $BCE^+ = BCEF$ and $EF^+ = EF$, E_3 does not

contain E_1. Hence $E_1 \bigcup E_3$ is an equivalent expression to compute the *ABCF*-total projection. □

Having showed how to test containment of two simple cje's efficiently, we are now ready to show how to minimize a simple cje. By Corollary 5.2, simple cje's can be optimized by methods in [ASU1][ASU2][ASSU][S3]. However, due to the restricted nature of this class of expressions, a simpler method is possible.

Algorithm 3: Minimizing a simple cje.

Input: A simple cje $E = \pi_X(R_{i_0} \bowtie \pi_{Y_1Z_1}(R_{i_1}) \bowtie \cdots \bowtie \pi_{Y_mZ_m}(R_{i_m}))$.

Output: E' - A minimal expression equivalent to E.

Method:

(1) Initialize three arrays of length m as follows, where m is the number of fd's in the ds. For $j = 1$ to m do:

$\text{TAG}[j] = R_{i_j}$.

$\text{SUBEXP}[j] = Y_jZ_j$.

$\text{CLOSURE}[j] = (Y_j)^+$.

(2) For j=1 to m do:

If there exists k, $k \neq j$, $1 \leq k \leq m$, such that $\text{TAG}[k] = \text{TAG}[j]$ and $\text{CLOSURE}[k] \supseteq \text{CLOSURE}[j]$ then $\text{SUBEXP}[k] = \text{SUBEXP}[k] \bigcup \text{SUBEXP}[j]$ and $\text{TAG}[j] = \emptyset$.

(3) Construct an expression T as follows. $T = R_{i_0}$.

For $j = 1$ to m do:

If $\text{TAG}[j] \neq \emptyset$ then $T = T \bowtie \pi_{SUBEXP[j]}(TAG[j])$.

(4) The final expression returned is $E' = \pi_X(T)$. □

Theorem 5.4: E' produced by Algorithm 3 is equivalent to E and is minimal in the number of join operations. The time complexity of Algorithm 3 is $O(m^*(|F|+|U|^* \log |U|)+m^*(m-1)^*|U|)$.

[*Proof*]: See [C]. □

Knowing how to optimize and test containment of simple cje's efficiently, we have an algorithm to optimize a union of simple cje's. We first eliminate redundant subexpressions (i.e. simple cje's) from the union using Theorems 5.3, 5.1 and 5.2. We then use Algorithm 3 to optimize each of the remaining simple cje's. Since each stage can be done efficiently, optimization of unions of simple cje's can be done efficiently. By a Theorem in [SY], the expression returned is minimal both in the number of subexpressions and the number of join operations.

Example 6: $\mathbf{R} = \{R_1(AB), R_2(ABCDEF)\}$. $F = \{AB \rightarrow D, BC \rightarrow E, B \rightarrow C, D \rightarrow F, E \rightarrow F\}$.

In Example 5, the expression to compute the *ABCF*-total projection is

$$E = E_1 \bigcup E_3 = \pi_{ABCF}(R_1 \bowtie \pi_{BC}(R_2) \bowtie \pi_{BCE}(R_2) \bowtie \pi_{EF}(R_2)) \bigcup \pi_{ABCF}(R_2).$$

Since E_3 cannot be minimized any more, let us consider E_1. $BC^+=BCEF$, $BCE^+=BCEF$ and $EF^+=EF$. Hence in step (2) of Algorithm 3, TAG[1] and TAG[3] are set to $\emptyset$ and SUBEXP[2] = *BCEF*. Hence $\pi_{ABCF}(R_1 \bowtie \pi_{BCEF}(R_2))$ is returned in step (4) of the Algorithm. Therefore an expression with the minimal number of subexpressions and join operations that computes the *ABCF*-total projection is $\pi_{ABCF}(R_1 \bowtie \pi_{BCEF}(R_2)) \bigcup \pi_{ABCF}(R_2)$. □

6. Conclusions

We proposed and defined cje's as a means to simulate the representative instance when fd's are given as constraints. This kind of expressions can be considered as a generalization of extension joins [H1]. We showed that for any $X \subseteq U$, the X-total projection of $CHASE_F(T_r)$ can be computed by a union of simple cje's when an independent scheme [GY] is assumed. This demonstrated an application of simple cje's. Also cje's can be used to compute the X-total projections with respect to a set of fd's F for other bounded schemes that are not independent, as illustrated in Examples 1 and 3. We also derived an efficient and

simple algorithm to optimize unions of simple cje's.

Even though Algorithm 3 returns an equivalent expression that is minimal in the number of join operations, the size of intermediate relations in the minimal expression may not be optimal. For instance in Example 6, since $B \rightarrow CF \in F^+$ and $BCF^+ = BCEF^+$, it is easy to verify that $\pi_{ABCF}(R_1 \bowtie \pi_{BCF}(R_2))$ is equivalent to $\pi_{ABCF}(R_1 \bowtie \pi_{BCEF}(R_2))$. It is interesting to find a systematic way to obtain such optimal expressions. Since computation of the X-total projections for some bounded schemes require complex cje's, as was illustrated in Example 3, it is natural that we should study the optimization of this class of expressions. Also it has been hypothesized that the full join dependency $\bowtie\mathbf{R}$ is essential in most applications [FMU]. Hence it is interesting to know if cje's can still be used to compute the X-total projections when the full join dependency $\bowtie\mathbf{R}$ is included as part of the constraints.

Recently several researchers have pointed out that in some cases the answer generated by the total projection approach may not be intuitively appealing [L][MUV][Y2]. They argued that instead of applying the query to the intersection of all weak instances, the query should be applied to each weak instance individually and then the intersection of the resulting answers taken. Generating the answer using this alternative approach is considered in [Y2], when the full join dependency $\bowtie\mathbf{R}$ is given as a constraint. It would be interesting to know if cje's can still be used to generate answers in this alternative approach when fd's are given as constraints.

Acknowledgement

The author would like to thank P. Atzeni, 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 the Natural Sciences and Engineering Research Council of Canada.

References

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

[AK] Aho, A.V., Kernighan, B.W., cited in [MRW].

[ASSU] Aho, A.V., Sagiv, Y., Szymanski, T.G., Ullman, J.D., "Inferring a Tree from Lowest Common Ancestors with an Application to the Optimization of Relational Expressions," *SIAM J. of Computing 10:3 (1981)*, pp.405-421.

[ASU1] Aho, A.V., Sagiv, Y., Ullman, J.D., "Equivalence of Relational Expressions," *SIAM J. of Computing 8:2 (1979)*, pp.218-246.

[ASU2] Aho, A.V., Sagiv, Y., Ullman, J.D., "Efficient Optimization of a Class of Relational Expressions," *ACM TODS 4:4 (1979)*, pp.435-454.

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

[BV] Beeri, C., Vardi, M., "The Implication Problem for Data Dependencies," *Proc. 8th Int. Conference on Automata, Languages, and Programming*, Lecture Notes in Computer Science, 115, Springer-Verlag, New York, 1981, pp.78-85.

[CK] Carlson, C.R., Kaplan, R.S., "A Generalized Access Path Model and Its Application to a Relational Data Base System," *Proc. SIGMOD 1976*, pp. 143-154.

[C] Chan, E.P.F., "Query Answering and Schema Analysis under the Weak Instance Model," forthcoming Ph.D. Thesis, University of Toronto, 1984.

[CM] Chan, E.P.F., Mendelzon, A.O., "Independent and Separable Database Schemes," *Proc. PODS 1983*, pp. 288-296. Full paper submitted to publication.

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

[F] Fagin, R., "Horn Clauses and Database Dependencies," *JACM 29:4 (1982)*, pp. 952-985.

[FMU] Fagin, R., Mendelzon, A.O., Ullman, J.D., "A Simplified Universal Relation Assumption and Its Properties," *ACM TODS 7:3 (1982)*, pp. 343-360.

[GM1] Graham, M.H., Mendelzon, A.O., "Strong Equivalence of Relational Expressions Under Dependencies," *IPL 14:2 (1982)*, pp. 57-62.

[GM2] Graham, M.H., Mendelzon, A.O., "On Total Projections Computable by Relational Algebra," unpublished manuscript, 1982.

[GMV] Graham, M.H., Mendelzon, A.O., Vardi, M.Y., "Notions of Dependency Satisfaction," *TR STAN-CS-83-979*, Stanford University, August, 1983.

[GY] Graham, M.H., Yannakakis, M., "Independent Database Schemas," *Proc. PODS 1982*, pp.199-204. Full paper to appear in *JCSS*.

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

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

[KU] Korth, H., Ullman, J.D., "System U - A Database System based on the Universal Relation Assumption," *Proc. XP1 Workshop, 1980.*

[L] Laver, K., "No-Information Nulls and Regions of the Universe," unpublished manuscript, 1983.

[MMS] Maier, D., Mendelzon, A.O., Sagiv, Y., "Testing Implications of Data Dependencies," *ACM TODS 4:4 (1979)*, pp.455-468.

[MRW] Maier, D., Rozenshtein, D., Warren, D.S., "Windows On The World," *Proc. SIGMOD 1983*, pp. 68-78.

[MU] Maier, D., Ullman, J.D., "Maximal Objects and the Semantics of Universal Relation Databases," *ACM TODS 8:1 (1983)*, pp.1-14.

[MUV] Maier, D., Ullman, J.D., Vardi, M.Y., "The Revenge of the JD," *Proc. PODS 1983*, pp. 279-287.

[MW] Maier, D., Warren, D.S., "Specifying Connections for a Universal Relation Scheme Database," *Proc. SIGMOD 1982*, pp.1-7.

[O] Osborn, S.L., "Towards a Universal Relation Interface," *Proc. VLDB 1979*, pp. 52-60.

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

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

[S3] Sagiv, Y., "Quadratic Algorithms for Minimizing Joins in Restricted Relational Expressions," *SIAM J. of Computing 12:2 (1983)*, pp. 316-328.

[SY] Sagiv, Y., Yannakakis, M., "Equivalence Among Relational Expressions with the Union and Difference Operators." *JACM 27:4 (Oct 1980)*, pp.633-655.

[SP] Schenk, K.L., Pinkert, J.R., "An Algorithm for Serving Multi-relational Queries," *Proc. SIGMOD 1977*, pp.10-19.

[U] Ullman, J.D., "The U.R. Strikes Back," *Proc. PODS 1982*, pp. 10-22.

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

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

[Y2] Yannakakis, M., "Querying Weak Instances," *Proc. PODS 1984*, pp.275-280.