数理解析研究所講究録
第 731 巻 1990 年 274-285

全称制約および存在制約を含む集合制約質問の処理法について

Processing Set Constraint Queries Including Universal and Existential Constraints

岩井原 瑞穂 (九大 工)
上林 弥彦 (九大 工)

1 Introduction

Recently in the database field, deductive databases and object oriented databases are two major areas which have attracted many researchers. For deductive databases, several logical query languages, such as LDL1 [1] and ELPS [7] were proposed. They can handle complex objects by introducing set-valued variables in logic programming.

On the other hand, the nested algebra [6], which has been a representative and basic complex object language, turns out to have less expressive power, since it cannot express the transitive closure [9]. To incorporate such fundamental operations for deduction, the power-set algebra was introduced [2], which is an extension of the nested algebra obtained by adding the power-set operator.

However, these approaches cause difficulties in processing queries. Those languages are in a sense too expressive, and queries often appear that require too much time to process. Actually, by increasing the nesting level of sets, the processing time becomes exponential time, double exponential time, in the size of databases [4]. It is therefore important to investigate classes of queries that can be computed in polynomial time in set-valued logical/algebraic query languages.

Set constraint queries produce set-valued solutions, which are subsets of given base relations, and query conditions are given as "constraints", which describe the solution sets using first-order quantified predicates on their member tuples.

In [5], we showed several subclasses of queries which can be computed in polynomial time, and other subclasses which are NP-time computable. For NP-complete queries, we considered the use of data dependencies, such as join dependencies (JDs) and functional dependencies (FDs), which are maintained in the database as semantic constraints. If certain data dependencies exist, by utilizing these dependencies we can process the queries in polynomial time.

In [5] we discussed a class of constraints called *binary constraints.* In this paper, we consider other classes of constraints, called *unary constraints.* Unary constraints include conditions such as each solution must contain at least one tuple that satisfies given conditions (*existential constraints*), and/or all tuples in a solution must satisfy given constraints (*universal constraints*). We show a processing method for unary constraints which requires polynomial time if binary constraints can be computed in polynomial time.

If we combine the method for unary constraints and the method utilizing JDs and FDs, a problem arises, since JDs and FDs are destroyed while processing unary constraints. For query constraints, we consider conditions which preserve JDs and FDs in order to apply the two methods successfully.

For the class of JDs and FDs, it is generally impossible to express all dependencies in a give relational expression accurately [3]. However, we will mainly be concerned with preserving dependencies, and will also discuss general relational algebra expressions including set difference and union operators.

2 Preliminaries

Relational database

For a given attribute set $X = \{A_1, A_2, \ldots, A_n\}$, a *relation* $R(X)$ on X is a finite set of mappings $\tau : \{A_1, A_2, \ldots, A_n\} \mapsto D_1 \times D_2 \times \cdots \times D_n$, where each D_i is the domain of attribute A_i, and each A_i is mapped to an element of D_i. Each mapping τ shall be called a *tuple* of $R(X)$. X shall be called a *relational scheme* of $R(X)$. In general, the relational scheme of a relation R shall be denoted by $\underline{R}$.

The relation R can be represented as a table which has attributes for rows, and tuples for columns. In this paper we use the first alphabet letters $A, B, C, \ldots$ for the names of single attributes, and the last alphabet letters $\ldots, X, Y, Z$ for the names of attribute sets. The union of two attribute sets may be simply denoted by the concatenation of their names.

Relational Algebra

Relational algebra is a procedural query language for relational databases. The six major operations of relational algebra are described as follows. The value for an attribute set X of a tuple μ is denoted by $\mu[X]$.

For relations $R_1(X)$ and $R_2(X)$ of the same scheme X, $R_1 - R_2$ and $R_1 \cup R_2$ are the *set difference* and *union* of tuple sets of R_1 and R_2, respectively.

For a relation $R(X)$ and an attribute set $Y (\subseteq X)$, $R[Y] = \{\mu[Y] \mid \mu \in R\}$ is called the *projection* of R onto Y.

Let θ be one of the comparison operators $=$, $>$, $\geq$, $<$, $\leq$, $\neq$. For θ, a constant c, a relation $R(X)$ and an attribute $A(\in X)$, $R[A\,\theta\,c] = \{\mu \mid \mu \in R, \mu[A]\,\theta\,c\}$ is called the *selection* of R on Y.

Let $R_1(X_1)$ and $R_2(X_2)$ be relations. The *(natural) join* of R_1 and R_2 shall be denoted by $R_1 \bowtie R_2 = R(X_1X_2)$, where $R(X_1X_2) = \{\mu \mid \mu[X_1] \in R_1, \mu[X_2] \in R_2\}$. The *Cartesian product* of R_1 and R_2 is denoted by $R_1 \times R_2$.

Tuple Relational Calculus (TRC)

Tuple relational calculus (TRC) is a declarative language for relational databases. TRC is defined as follows.

For a predicate symbol p, $p(\tau)$ is an atomic formula which states that τ is in the relation of p. $X\theta Y$ is an atomic formula, where each X and Y is either a constant or a value of a tuple variable, and θ is an arithmetic comparison operator. An atomic formula is a TRC expression.

For the TRC expressions f_1 and f_2, and the tuple variable τ which appears freely in f_1 and f_2, then $f_1 \wedge f_2$, $f_1 \vee f_2$, $\neg f_1$, $(\exists\tau)f_1$, $(\forall\tau)f_1$ are all TRC expressions.

For the TRC expression f with unique free variable τ, the relation value which f defines is $F = \{\tau \mid f(\tau)\}$.

For each 'safe' TRC expression, there exists a relational algebra expression which defines the same relation [10]. If a TRC expression f is not safe, f may define a relation of infinite size.

Functional Dependencies and Join Dependencies

For a relation $R(X)$, R is said to satisfy the *functional dependency (FD)* $X_1 \rightarrow X_2$ $(X_1, X_2 \subseteq X)$, if for any two tuples μ_1 and μ_2 of R, $\mu_1[X_1] = \mu_2[X_1]$ implies $\mu_1[X_2] = \mu_2[X_2]$.

A relation R is said to satisfy a *join dependency (JD)* $\bowtie[X_1, X_2, \ldots, X_n]$, if $R = R[X_1] \bowtie R[X_2] \bowtie \cdots \bowtie R[X_n]$ holds. Here we call each X_i a *component* of the JD.

3 Set constraint query

Definition 1 : Let R be a relation of the scheme $\underline{R}$. The *query constraint* $C(S)$ is a condition for relations S that are subsets of R. Three types of query constraints are described as follows:

$$\begin{aligned} Cb(S) &= \bigwedge_{i=1}^{h} (\forall\mu \in S)(\forall\nu \in S) b_i(\mu,\nu) \quad \textit{(binary constraints)}, \\ Ca(S) &= \bigwedge_{i=1}^{l} (\forall\mu \in S) a_i(\mu) \qquad\qquad \textit{(universal constraints)}, \end{aligned}$$

$$Ce(S) \;=\; \bigwedge_{i=1}^{k} (\exists\mu \in S) e_i(\mu) \qquad (\textit{existential constraints}),$$

where μ and ν are tuple variables on R. R is called a *base relation*. Each $b_i(\mu,\nu)$ is a TRC expression upon free variables μ and ν. Each $a_i(\mu)$ and $e_i(\mu)$ is a safe TRC expression upon a free variable μ. Existential and universal constraints are also called *unary constraints*.

For query constraints $C(S) = Cb(S) \wedge Ca(S) \wedge Ce(S)$, the following function $Q_C(R)$ defined below is a set constraint query on $\underline{R}$.

$$Q_C(R) = \{S \mid S \subset R \wedge C(S)\}$$

Each $S \in Q_C(R)$ is called a *solution* of $Q_C(R)$. □

Note that in the above definition for Cb and Ca, the conjunctions of universally quantified clauses can be transformed into single universally-quantified clauses.

For the predicate $b(\mu,\nu)$ of a binary constraint, since each pair μ and ν of tuples in each solution S must satisfy $b(\mu,\nu)$, $b(\mu,\nu)$ and $b(\mu,\mu)$, we can assume that $b(\mu,\nu)$ is symmetric and reflexive without loss of generality. Thus we can define an undirected graph for the binary predicate $b(\mu,\nu)$ as follows.

Definition 2 : For the predicate $b(\mu,\tau)$ of a binary constraint Cb and a base relation R, let $B(R,b)$ be an undirected graph, called a *tuple graph for R and b*, obtained as follows.

The node with label μ corresponds to a tuple μ of R, and there is an edge between two nodes μ and τ in $B(R,b)$ if and only if $b(\tau,\mu)$ holds. □

For a set constraint query $Q_{Cb}(R)$ consisting a binary constraint Cb, each solution S of $Q_{Cb}(R)$ corresponds to a clique of the tuple graph $B(R,b)$, since each pair of tuples in S must satisfy b.

An undirected graph of n nodes may have up to 2^n cliques, thus a set constraint query may have an exponential number of solutions. It is not practical to generate all solutions.

In the following, we consider combinatorial problems which are described by set constraint queries, where the optimal solutions correspond to maximum cardinality solutions of set constraint queries. We call maximum cardinality solutions simply maximum solutions, and we consider algorithms which produce a maximum solution. Note that for a query there are solutions of the same maximum cardinality, and algorithms will choose one of the maximum solutions.

There are two types of complexity measures for database queries. For a set constraint query $Q_C(R)$, if $S \in Q_C(R)$ can be decided in time (or space) bounded by a function f of the total size of relations on which C is defined, then $Q_C(R)$ is said to have time (or space) *data complexity* f. Alternatively, if f is a function of the length of expression C, then $Q_C(R)$ has *expression complexity* f [11]. In this paper, we will consider only the data complexity measure.

Example 1 : Let us consider the following job assignment problem. Let $R_1(EJT)$ be a relation consisting of attribute E for employee names, attribute T for time slots, and attribute J for types of jobs. A tuple of R_1 means that a certain employee can engage in a certain type of job at a certain time. A job assignment problem is finding a subset S of R_1 which satisfies some given constraints. Subset S is optimal if it is an assignment of the greatest possible number of jobs to employees. Figure 1 shows an instance of $R_1(EJT)$.

The constraint is that each employee can engage in only one type of job at the same time slot. This constraint is described by the following binary constraint:

$$\begin{aligned} Cb_1(S) \quad = \quad & (\forall \mu \in S)(\forall \nu \in S)(R_1(\mu) \wedge R_1(\nu) \wedge \\ & (\neg(\mu[E] = \nu[E] \wedge \mu[T] = \nu[T]) \vee (\mu[J] = \nu[J]))) \end{aligned}$$

□

E	J	T
Mike	cashier	A
Mike	cashier	B
Mike	waiter	A
Tom	cashier	A
Tom	kitchen	B
Tom	waiter	C
Tom	waiter	D
John	cashier	B
John	waiter	D

Figure 1: An instance of $R_1(EJT)$

The above binary constraint Cb_1 is equivalent to an FD $ET \rightarrow J$. Thus we can use FD sets to describe query constraints.

Definition 3 : A set constraint query whose binary constraints can be expressed as a set of FDs, is called an *FD* set constraint query. □

4 Processing unary constraints

Example 2 : In addition to the constraints of Example 1, we assume that the assignment must include at least one female employee. This constraint can be described by the following existential constraint Ce_1:

$$\begin{aligned} Ce_1(S) &= (\exists\mu \in S)(R_1(\mu) \wedge \\ &\qquad (\exists\nu)(R_2(\nu) \wedge \mu[E] = \nu[E] \wedge \nu[M] = \text{“female”})), \end{aligned}$$

where $R_2(EMA)$ is a relation which contains tuples such that employee E has sex M and age A. The query for the above condition is described as $Q_{Cb1 \wedge Ce1}(R_1)$. $\square$

In the following, we show an algorithm which solves a query of the form $Q_{Cb \wedge Cu}$ consisting of the binary constraints Cb and the unary constraints Cu, by first converting it into a set of queries of the form Q_{Cb} which do not include the unary constraints Cu.

Every unary constraint Cu can be described as the conjunction of universal constraint Ca and existential constraint Ce.

$$Cu(S) = Ca(S) \wedge Ce(S),$$

where

$$Ca(S) = (\forall\mu \in S)g(\mu), \quad Ce(S) = \bigwedge_{i=1}^{k}(\exists\mu \in S)f_i(\mu).$$

Algorithm 1 : Evaluation of set constraint queries including unary constraints

Input: A set constraint query $Q_{Cb \wedge Cu}(R)$, where Cb is a binary constraint and Cu is a unary constraint, and R is a base relation.

Output: A maximum solution of $Q_{Cb \wedge Cu}(R)$.

Method: We assume that this algorithm can call some Algorithm α that produces one maximum solution of $Q_{Cb}(R)$.

(1) Evaluate the predicates $f_1, \ldots, f_k$ and g into the finite relations $F_1, \cdots, F_k$ and G, respectively. Note that the tuples which should be tested whether satisfying those predicates can be limited to the base relation. Therefore the relations $F_1, \cdots, F_k$ and G become finite, and subsets of the base relation R. Each solution must be a subset of G to satisfy the universal constraint.

(2) Construct the tuple graph $B(G, b)$ from G and b.

(3) Find all tuple sets $q_1, \ldots, q_m$ such that for every $q_j = \{\mu_1, \ldots, \mu_k\}$ (μ_i's need not be distinct), each μ_i is taken from F_i, and each pair of tuples μ_i and μ_l satisfies $b(\mu_i, \mu_l)$. Each q_j thus corresponds to a clique of $B(G, b)$. Each solution of $Q_{Cb \wedge Cu}(R)$ must contain q_i for some i to satisfy the existential constraints Ce.

(4) For each clique q_i, $i = 1, \ldots, m$, let W_i be the relation obtained by removing all tuples τ such that there exists a tuple μ in q_i for which $b(\tau, \mu)$ does not hold.

(5) For each W_i, $i = 1, \ldots, m$, find a maximum solution S_i of $Q_{Cb}(W_i)$ by applying Algorithm α for Q_{Cb}. Report one of the maximum S_i's. $\square$

Theorem 1 : *For a set constraint query on $\underline{R}$ consisting of a binary constraint Cb, if a maximum solution of $Q_{Cb}(R)$ can be computed in polynomial time for any base relation R, the query $Q_{Cb \wedge Cu}(R)$ consisting of Cb and an arbitrary unary constraint Cu can also be computed in polynomial time.*

Proof: It is easily seen that the maximum S_i in Step 5 of Algorithm 1 satisfies all constraints Cb, Ca and Cu, and maximum in the solutions of $Q_{Cb \wedge Cu}(R)$.

We now show that the computation may be performed in polynomial time. Let n be the number of tuples in the base relation R, denoted by $|R|$.

Since every relational algebra expression can be evaluated in polynomial time [11], and since $|F_1| \leq n, \cdots, |F_k| \leq n$, and $|G| \leq n$, the relations $F_1, \ldots, F_k$, and G may all be computed in polynomial time.

Step 2 can be computed in polynomial time, because the binary predicate $b(\tau, \mu)$ defines a relation which is a subset of $G \times G$ corresponding to the edge set of $B(G, b)$. In Step 3, since each clique q_i corresponds to an element of the Cartesian product $F_1 \times \cdots \times F_k$, the number of cliques, m, is no more than n^k. $|G| \leq n$ implies that $|W_i| \leq n$ for $1 \leq i \leq m$. From the hypothesis, a maximum solution of $Q_{Cb}(W_i)$ can be computed in polynomial time, that is, $O(n^p)$ time for some positive constant p. Then Step 5 can be computed in $O(n^p m)$ time, hence $O(n^{p+k})$ time. Here k, the number of predicates in the existential constraint, is independent of n. $\square$

Note that in Step 3 of Algorithm 1, if no clique q_i is generated, then there is no solution for $Q_{Cb \wedge Cu}(R)$. In this case, we can abort the computation of solutions before evaluating Q_{Cb} in Step 5. On the other hand, if there exists a non-empty q_i, it is a solution for $Q_{Cb \wedge Cu}(R)$.

5 Processing queries utilizing database dependencies

In FD set constraint queries, FDs are used as the query constraints. However, FDs have been used to express semantic constraints and are often maintained in database management systems. Since query conditions and data dependencies can be compared of the same level, we can take advantage of these data dependencies for query processing. In this section, we give an example showing that an NP-complete query for arbitrary input base relations can be processed in polynomial time if certain dependencies such as FDs and JDs exist in the base relation. More detailed query processing methods utilizing data dependencies are discussed in [5].

Example 3 : In the job assignment problem of the previous examples, let us suppose that each employee is assigned only one type of job, and each type of job requires only one time slot and only one employee. This constraint is equivalent to computing a one-to-one correspondence between the values of E, T and J. This correspondence is denoted by the FD binary constraint:

$$C_{b2} = \{E \rightarrow J, J \rightarrow T, T \rightarrow E\} \qquad \square$$

Proposition 1 : *The problem of deciding whether $Q_{Cb_2}(R_1)$ has a solution of n tuples is NP-complete.*

Proof :(sketch) By a reduction of the 3D-MATCHING problem [8] to $Q_{Cb_2}(R_1)$. $\square$

If the base relation $R_1(ETJ)$ satisfies certain data dependencies, Proposition 1 fails if P$\neq$NP. For example, if all employees care only about the types of jobs and don't care about the time slots, R_1 satisfies $\bowtie[EJ, JT]$, and can be decomposed into $R_1[EJ]$ and $R_1[JT]$. In this case, we can compute a maximum solution of $Q_{Cb_2}(R_1)$ by constructing a network from $R_1[EJ]$ and $R_1[JT]$, and then applying the maximum network flow algorithm to this network [5].

6 Changes of data dependencies in the presence of unary constraints

In the previous section, we observed that if certain dependencies exist, there is considerable reduction in the computational complexity of queries. However, by adding unary constraints to queries, the required data dependencies of base relations may be destroyed.

Example 4 : In addition to the constraint Cb_2 in Example 3, consider the following constraint. Let us suppose that the employees younger than 20 must not work in the midnight time slot

"D". This condition is denoted by the following universal constraint Ca_2:

$$Ca_2(S) = (\forall \mu \in S) g_1(\mu),$$

where

$$g_1(\mu) = (R_1(\mu) \ \wedge \ \neg((\exists \nu)(R_2(\nu) \\ \wedge \mu[E] = \nu[E] \wedge \mu[T] = \text{"D"} \wedge \nu[A] < 20))).$$

The relation of the TRC expression g_1, namely $G_1 = \{\mu \mid g_1(\mu)\}$, is transformed into the following relational algebra expression:

$$G_1 = R_1 - (R_1[T = \text{"D"}] \bowtie R_2[A < 20])[ETJ].$$

The above expression contains a set difference. The relation G_1 does not satisfy $\bowtie[EJ, JT]$ while R_1 does. Consider the instances provided in Figure 2. Figure 2-(a) shows an instance of R_1 that satisfies $\bowtie[EJ, JT]$. Figure 2-(b) shows an instance of $R_2(EMA)$. Figure 2-(c) shows the relation G_1 produced by the above relational expression. Since G_1 does not contain the tuple (Tom, kitchen, D), G_1 does not satisfy $\bowtie[EJ, JT]$. □

E	J	T
Tom	kitchen	C
Tom	kitchen	D
John	kitchen	C
John	kitchen	D

(a) $R_1(EJT)$

E	M	A
Tom	male	18
John	male	24

(b) $R_2(EMA)$

E	J	T
Tom	kitchen	C
John	kitchen	C
John	kitchen	D

(c) $G_1(EJT)$

Figure 2: Destruction of a join dependency

In the case where the data dependencies used by some algorithm α to satisfy binary constraints are destroyed while processing unary constraints, as Steps 1–4 of Algorithm 1, we cannot use Algorithm α at Step 5 anymore. Furthermore, there are cases where a query which is polynomial-time computable when certain dependencies exist, becomes NP-complete when unary constraints are added.

Suppose that a base relation satisfies a set D of data dependencies, and that a query Q_{Cb} of a binary constraint Cb can be processed by an algorithm α which presupposes the existence of D. If each W_i in Step 4 satisfies D, then we can use Algorithm α at Step 5 without changing Algorithm 1. Therefore, we must specify conditions for binary and unary constraints which preserve data dependencies in the steps of Algorithm 1.

7 Query constraints preserving dependency sets

In this section, we discuss sufficient conditions for binary and unary constraints which guarantee that unary constraints may be processed in polynomial time, by showing that a set D of JDs and FDs is preserved during the execution of Algorithm 1.

7.1 FD sets

Lemma 1 : *If a relation $R(Z)$ satisfies an FD $X \rightarrow Y$, $(X, Y \subset Z)$, then any $S \subset R$ satisfies $X \rightarrow Y$.*

Proof: Suppose that S does not satisfy $X \rightarrow Y$. Then there exist tuples $\mu, \nu \in S$ such that $\mu[X] = \nu[X]$ and $\mu[Y] \neq \nu[Y]$. Since $\mu, \nu \in R$, R does not satisfy $X \rightarrow Y$, which is a contradiction. □

Each W_i in Algorithm 1 is a subset of the base relation R. By Lemma 1, each W_i satisfies all FDs which R satisfies.

7.2 JD sets

For a TRC expression $f(\mu)$ on $\underline{R}$ that has μ as a free variable, let $att(f)\,(\subseteq \underline{R})$ be the union of the attributes of μ appearing in formulas of the form $X\theta Y$ in f. For instance, for g_1 of Example 4, $att(g_1) = ET$.

Lemma 2 : *Suppose that a relation R satisfies a JD $j = \bowtie [X_1, \ldots, X_m]$. For a safe TRC expression $f(\mu)$ on $\underline{R}$, if there exists a component X_h of j such that $att(f) \subseteq X_h$, then for $F = \{\mu \mid f(\mu)\}$,*

$$F = F[X_h] \bowtie R[\cup_{i \neq h} X_i].$$

Proof: Suppose that $\mu \in F$. Then $\mu[X_h] \in F[X_h]$. Since $F \subseteq R$, we have $F[X_h] \subseteq R[X_h]$, and $\mu \in \{\mu[X_h]\} \bowtie R[\cup_{i \neq h} X_i] \subseteq F[X_h] \bowtie R[\cup_{i \neq h} X_i]$.

Suppose that $\mu \in F[X_h] \bowtie R[\cup_{i \neq h} X_i]$. Since j implies $\bowtie[X_h, \cup_{i \neq h} X_i]$, we have $\mu \in R$. The attributes of μ that appear in $f(\mu)$ are included in X_h, and $\mu[X_h] \in F[X_h]$, hence the values of μ satisfy f. Therefore $\mu \in F$. $\square$

Lemma 3 : *For a safe TRC expression $f(\mu)$ on $\underline{R}$, suppose that $f(\mu) = f_1(\mu) \wedge \cdots \wedge f_k(\mu)$. Also suppose that a relation R satisfies $j = \bowtie[X_1, \ldots, X_m]$. For $1 \leq i \leq k$, if there exists a component $X_{p(i)}$ $(1 \leq p(i) \leq m)$ of j such that $att(f_i) \subseteq X_{p(i)}$, then $F = \{\mu \mid f(\mu)\}$ satisfies j.*

Proof: Let $F_i = \{\mu \mid f_i(\mu)\}$, then $F_i \subseteq R$ and $F = \cap_i F_i$. If F does not satisfy j, we can assume that there exist m (not necessarily distinct) tuples $\nu_1, \ldots, \nu_m$ in F, and that there does not exist a tuple ν in F such that $\nu[X_1] = \nu_1[X_1], \ldots, \nu[X_m] = \nu_m[X_m]$. Then for some h, $\nu \notin F_h$ and $\nu[X_{p(h)}] = \nu_{p(h)}[X_{p(h)}] \in F_h[X_{p(h)}]$. By Lemma 2, $F_h = F_h[X_{p(h)}] \bowtie R[\cup_{i \neq p(h)} X_i]$, hence $\nu \in \{\nu[X_{p(h)}]\} \bowtie R[\cup_{i \neq p(h)}] \subseteq F_h$. Thus $\nu \in F_h$, a contradiction. $\square$

Henceforth, we will say that a TRC expression $f(\mu)$ is *consistent* with j if it satisfies the condition of Lemma 3.

7.3 Sufficient conditions for preserving dependency sets

Theorem 2 : *Suppose that for a set constraint query Q_{Cb} on $\underline{R}$ consisting of a binary constraint Cb, and that a maximum solution of $Q_{Cb}(R)$ can be computed in polynomial time if R satisfies a set D of JDs and FDs. Then, for an arbitrary universal constraint Ca whose TRC expression on $\underline{R}$ is consistent with each JD of D, a maximum solution of the query $Q_{Cb \wedge Ca}$ can be computed in polynomial time.*

Proof: The maximum solution of the query $Q_{Cb \wedge Ca}(R)$ can be computed by Algorithm 1, since by Lemma 1 and Lemma 3, the relation G in Step 1 satisfies all the dependencies of D. $\square$

Theorem 3 : *For $Q_{Cb}(R)$ and D defined in Theorem 2, consider the predicate $b(\mu, c)$ obtained by substituting a variable ν of the binary predicate $b(\mu, \nu)$ of Cb by a constant tuple c. If $b(\mu, c)$ is consistent with each JD of D, then for an arbitrary existential constraint Ce on $\underline{R}$, a maximum solution of the query $Q_{Cb \wedge Ce}(R)$ can be computed in polynomial time.*

Proof: By Theorem 1, it is sufficient to show that each W_i in Step 4 of Algorithm 1 satisfies D. By Lemma 1, W_i satisfies each FD in D. For each W_i, let $q_i = \{c_1, \ldots, c_k\}$, where the c_j's are constant tuples, not necessarily distinct. Then, W_i can be written

$$W_i = \{\mu \mid b(\mu, c_1) \wedge \cdots \wedge b(\mu, c_k)\}.$$

Note that $b(\mu, \nu)$ is reflexive and symmetric. The above expression is consistent with each JD of D by Lemma 3 and the hypothesis of the theorem. Therefore W_i satisfies each JD of D. □

Theorem 4 : *For $Q_{Cb}(R)$ and D of Theorem 2, suppose that the binary constraint Cb is equivalent to a set F of FDs. For each $X \rightarrow Y$ in F and each JD j in D, if $X \cup Y$ is contained in a component of j, then for an arbitrary existential constraint Ce on $\underline{R}$, a maximum solution of the query $Q_{Cb \wedge Ce}(R)$ can be computed in polynomial time.*

Proof: Since the binary predicate of Cb can be denoted as a conjunction of clauses corresponding to FDs of F, Cb satisfies the condition of Theorem 3. □

References

[1] Beeri, C., Naqvi, S., Ramakrishnan, R., Shmueli, O. and Tsur, S., "Sets and Negation in a Logic Database Language (LDL1)", ACM PODS 1987, pp. 21-37.

[2] Gyssens, M. and Gucht, D. V., " The Power Set Algebra as a Result of Adding Programming Constructs to the Nested Relational Algebra", ACM SIGMOD 1988, pp. 225-232.

[3] Hull R., "Finitely Specifiable Implicational Dependency Families", JACM Vol. 31, No. 2, pp. 210-226, 1984.

[4] Hull, R., Su, J., "On the Expressive Power of Database Queries with Intermediate Types", ACM PODS 1988, pp. 39-51.

[5] Iwaihara M. and Kambayashi, Y., "Processing Set Constraint Queries utilizing Data Dependencies", IPSJ Tech. Rep. 89-AL-12-39, pp. 269-276, 1989.

[6] Jaeschke, G. and Schek, H-J., "Remarks on the Algebra of Non-First-Normal-Form Relations", ACM PODS 1982, pp. 124-138.

[7] Kuper, G. M., "Logic Programming with Sets", ACM PODS 1987, pp. 11-20.

[8] Papadimitriou, C. H. and Steiglitz, K., "*Combinational Optimization*", Prentice-Hall, 1982.

[9] Paredaens, J., "Possibilities and Limitations of Using Flat Operators in Nested Algebra Expressions", ACM PODS 1988, pp. 29-38.

[10] Ullman, J. D., "*Principles of Database and Knowledge-Base Systems,*" Volume I, Computer Science Press, 1988.

[11] Vardi, M. Y., "The Complexity of Relational Query Languages", ACM STOC 1982, pp. 137-146.