数理解析研究所講究録
第 461 巻 1982 年 49-78

CONSTRUCTION OF SBDS-F3 : A RELATIONAL DATABASE WITH INFERENCE MECHANISM

Yosihisa Udagawa
and Setsuo Ohsuga

Institute of interdiscriplinary research
faculity of engineering, Tokyo University
4-6-1 Meguro-ku, Komaba, Tokyo 153, Japan

1. Introduction

In this paper, we define the multi-layer logic for relational databases, an extended many-sorted logic, and discuss implementation of a relational database system based on this logic. The system is termed SBDS-F3 (Structure Based Deduction System-Fortran version 3).

Query languages based on the predicate calculus have several advantages ,e.g.

(1) permit a user to request the data by its values ;

(2) provide a useful way to derive the facts derivable by using general axioms together with the facts stored explicitly in a database ;

(3) allow the database system to optimize execution of queries .

However, it is pointed out that this class of languages suffer from lack of expressive power. Query features for relational databases are categorized as follows.

(1) mapping : data values of a specified attribute associated

with a known data value for other attribute ;

(2) selection : entire record values associated with a specified key value ;

(3) projection : data values of a specified attribute, domain or relation ;

(4) restriction : data values that satisfy given conditions ;

(5) universal quantification : data values that corresponding to all the values of a specified attribute or domain ;

(6) arithmetic function : e.g. +, −, *, ÷ ;

(7) aggregation function : e.g. average, minimum, maximum ;

(8) group by : grouping of data values with a common domain value ;

(9) composition : composition of (1) to (8).

In the first order predicate logic (whether one-sorted or many-sorted), domain(s) of variable(s) and/or constants are fixed during the interpretation of formulas. This means that only domain-element relations can be described in this logic. Thus, queries classified into (7), (8) and (9) are difficult to be described by languages based upon the first order predicate logic. So far , various approaches have been proposed for extending the expressive power of query languages based on the predicate logic. For example, E.F.Codd [5] used built-in functions to manipulate functional operations. C.L.Chang [6] proposed to introduce numerical quantifiers , special symbols with the ad hoc meanings and procedural elements. S.Kunihuji [7] proposes the second-order m-sorted logic ($m \geq 0$), and formulate the systax and the semantics of his logic by model theoretic approach.

But, as far as the authors are aware, they do not provide systematic solution for the problem. The multi-layer logic discussed here is a logic obtained by introducing a method of structuring types, i.e set, into the many-sorted logic. As the result, this logic has enough descriptive power to express practical queries which contain functions whose arguments are grouped by some other variables and/or which involve nested aggregation functions. This extension has some analogy to the introduction of methods of combining types in a general-purpose programming language.

In section 2, we define the multi-layer logic for the relational database and prove some properties of this logic. Section 3 discusses how to describe a query against the relational database with a formula in the multi-layer logic. Section 4 concerns with virtual relations and inference algorithm for a query involving them. Implementation of the algorithm is also referred. In section 5, we discuss the algorithm which reduces a formula in the multi-layer logic into a sequence of database access procedure.

2. Definition of the multi-layer logic

2.1. Syntax of the multi-layer logic

Definition. 1. The alphabet of the multi-layer logic for a relational database query language is composed of the following :

(1) variable symbols : x,y,...,X,Y,...;

(2) function symbols : f, g ,...;

(3) predicate symbols : P, Q ,...;

(4) logical connectives and quantifiers : $\sim$,& ,V ,$\rightarrow$,$\equiv$,$\exists$,$\forall$;

(5) punctuations : /, [,], ", (,), comma ;

(6) truth symbols : T, F,

(7) domain symbols : I, S ,... ;

(8) set operator symbol : * .

In the multi-layer logic , as is the case of the many-sorted logic, it is assumed that there is a non-empty set I whose elements are called base sorts. For each base sort i, there is a non-empty set D_i.

Definition 2. A set J(I) of sort having a base sort set I, is defined recursively as follows.

(1) If $i \in I$,then $i \in J(I)$.

(2) If $i \in J(I)$, then $2^i - \{\emptyset\} \in J(I)$.

Definition 3. The sort of powerset *S of a domain S of sort j excluding the empty-set is $2^i - \{\emptyset\}$.

Example. 1. If S = {A, B}, then *S = { {A}, {B}, {A,B} } and **S = { {{A}}, {{B}}, {{A,B}}, {{A},{B}}, {{A},{A,B}}, {{B},{A,B}}, {{A},{B},{A,B}} }. If S is a domain of sort j, then *S and **S are domains of sorts $2^j - \{\emptyset\}$ and $2^{2^j - \{\emptyset\}} - \{\emptyset\}$.

Definition. 4. Base elements of a domain are elements whose internal structure is ignored except for the purpose of identifying them.

Definition. 5. Level of a base element is defined as 0 , and level of a set of base elements as 1. If a level of a set S is i, then a level of a set *S is i+1.

Example 2. {{-3,4},{0,8,2}} is a set of level 2, and base elements are -3, 4, 0, 8, 2.

Definition. 6. A term is defined inductively as follows :

(1) a constant of sort $j \in J(I)$ is a term of sort j ;

(2) a variable defined over a domain of sort $j \in J(I)$ is a term of sort j ;

(3) if $t_1,\dots,t_n$ denote terms of sorts $j_1,\dots,j_n$ and f is a n-place function symbol from sorts $j_1,\dots,j_n$ to j_f, then $f(t_1,\dots,t_n)$ is a term of sort j_f, where $n \geq 0$.

Definition. 7. If $x_0,x_1,\dots,x_n$ $(n \geq 0)$ are variables defined over S, *S, ..., $*_n\dots*_1S$, respectively, then $(Q_nx_n/*_n\dots*_1S)$ $(Q_{n-1}x_{n-1}/x_n)\dots(Q_0x_0/x_1)$ is a prefix in the multi-layer logic, where Q_i $(0\leq i\leq n)$ is either $\forall$ or $\exists$. When n=0 , a prefix is (Q_0x_0/S), which is the prefix in the many-sorted logic.

Definition. 8. A formula in the multi-layer logic is defined recursively as follows :

(1) if $t_1,\dots t_n$ are terms and P denotes a n-place predicate symbol, then $P(t_1,\dots,t_n)$ is a (atomic) formula, where $n\geq 0$.

(2) if A and B are formulas, then so are $(\sim A)$, $(A \,\&\, B)$, $(A \vee B)$, $(A \rightarrow B)$, $(A \equiv B)$;

(3) if $P(x_1,\dots,x_n)$ is a formula containing $x_1,\dots,x_n$ as free variables and $x_1,\dots,x_n$ are defined over domains $S_1,\dots,S_n$,

respectively, then $(Q_{i_1} x_{i_1}/S_{i_1})\ldots(Q_{i_n} x_{i_n}/S_{i_n})P(x_1,\ldots,x_n)$ is a formula, where $n \geq 0$ and $i_1,\ldots,i_n \in \{1,\ldots n\}$.

(4) formulas are generated only by a finite number of application of (1),(2),(3).

Example. 3. The formulas given below are formulas in the multi-layer logic.

$(\exists X/{*}N)(\forall x/X)(P(X, a/{*}M) \,\&\, Q(x))$;

$(\forall m/M)(\exists J/{*}N)(\forall j/J)(((\exists k/K)\ Q(a/I, f(J)) \,\&\, P(m, j, k)) \rightarrow R(m, j))$.

2.2. Semantics of the multi-layer logic

Definition 9. A structure σ is as follows.

(1) a non-empty set I of base sorts ;

(2) domains S_j of each sort j, where $j \in J(I)$;

(3) distinguished elements of S_j ;

(4) a set of n-place functions from $X\, S_{j_i}$ to S_f, where $i = 1,\ldots,n$, $j_i \in J(I)$, $f \in J(I)$;

(5) a set of n-place predicates over the domains $X\, S_{j_i}$.

Definition. 10. An assignment φ from a formula G to the structure is a mapping that satisfies the following conditions :

(1) to each free variable in a formula G, we assign an element of S_j, $j \in J(I)$;

(2) to each domain symbol in a formula G, we assign a domain S_j, $j \in J(I)$;

(3) to each constant in a formula G is assigned to some particular element of S_j, $j \in J(I)$;

(4) to each n-place function symbol f in a formula G, we assign a n-place function from $X\, S_{j_i}$ to S_f, where i = 1,...,n , $j_i \in J(I)$, $f \in J(I)$ (if n = 0, then we assign some particular element of S.) ;

(5) to each n-place predicate symbol P in a formula G, we assign a n-place predicate defined over the domain $X\, S_{j_i}$ (if n = 0 , then we assign either T or F.).

<u>Definition. 11.</u> An assignment φ is similar mod $x_1,\dots,x_n$ to φ' if the assignments is same except for the elements assigned to $x_1,\dots x_n$.

<u>Definition. 12.</u> An interpretation of a formula G is an ordered pair (σ, φ), where σ is a structure for a given formula G and φ is an assignment from G to σ.

<u>Definition 13.</u> The valuation of a formula G over the interpretation (σ, φ) , i.e. val(G, σ, φ), is inductively defined as follows.

(1) If G is a propositional letter , then val(G, σ, φ) = T iff φ(G) = T .

(2) If G is a n-place atom P($t_1,\dots,t_n$), then val(G, σ, φ) = T iff ($\varphi(t_1),\dots,\varphi(t_n)$) $\in \varphi$($P(t_1,\dots,t_n)$) .

(3) Let A and B are formulas. If G is of form ~A, (A & B), (A V B), (A $\rightarrow$ B) and (A $\equiv$ B), then val(G, σ, φ) = T iff the truth table (same as classical logic) for φ(A) and φ(B) yield T .

(4) If G is of form ($\exists$ x/S) H(x), then val(G, σ, φ) = T iff

there exists an assignment φ' which is similar mod x to φ and val(H, σ, φ) = T. That is, $(\exists\ x/S)\ H(x) = T$ iff there is at least one element $e \in S$ such that $H(e) = T$.

(5) If G is of form $(\forall\ x/S)\ H(x)$, then val(G, σ, φ) = T iff for all assignments φ' which is similar mod x to φ and val (H, σ, φ) = T. That is, $(\forall\ x/S)\ H(x) = T$ iff for all elements $e \in S$, $H(e) = T$.

(6) If val(G, σ, φ) $\neq$ T , then val(G, σ, φ) = F.

<u>Definition 14.</u> EQ(X,Y) : This predicate is defined over two sets of level m ($m \geq 0$) X and Y, whose base elements are ordered. For m = 0, EQ(X, Y) = T iff X = Y.
For m > 0, EQ(X, Y) = T iff for pairs of < x, y >, which are level m-1 sets of < X ,Y >, EQ(x, y) = T.

In the same manner NE(X,Y), GT(X,Y), etc. are defined.

<u>Definition 15.</u> LET(X,C) : LET(X,C) is semantically equivalent to $(\exists\ X/{*}D)$ EQ(X,C), where D is a set of level m ($m \geq 0$) and its sort includes that of C.

<u>Example. 4.</u> The interpretation of the formula $(\exists\ X/{*}N)(\forall\ x/X)$ (P(X, a/*M) & Q(x)) over the σ and φ below is as follows.

σ = ({ m, n } ,
$\{ \{ -1, 1, 2, 3 \}_m , \{ -2, -1, 0, 1, 2, 3 \}_n \}$,
$\{ \{ -1, 1, 3 \}_m \}$,
{} ,
{ $\lambda s \lambda t.s \subseteq t$, $\lambda s.s>0$ })

φ = ({} ,

$\{ M \rightarrow \{ -1, 1, 2, 3 \}_m , N \rightarrow \{ -2, -1, 0, 1, 2, 3 \}_n \}$,

$\{ a \rightarrow \{ -1, 1, 3 \}_m \}$,

{} ,

$\{ P \rightarrow \lambda s \lambda t . s \subseteq t , Q \rightarrow \lambda s . s>0 \}$)

$\mathrm{val}((\exists X/{*}N)(\forall x/X) (P(X, a/{*}M) \& Q(x)) , \sigma , \varphi)$

$= \mathrm{val}((\exists X/{*}\{ -2, -1, 0, 1, 2, 3 \}_n)(\forall x/X) (\lambda X . X \subseteq$

$\{ -1, 1, 3 \}_m \& \lambda x . x>0) , \sigma , \varphi)$

$= \mathrm{val}(\forall x/\{-2\}_n (\{-2\}_n \subseteq \{ -1, 1, 3 \}_m \& x>0)$

$\vee \forall x/\{-1\}_n (\{-1\}_n \subseteq \{ -1, 1, 3 \}_m \& x>0)$

$\vee \forall x/\{ 0\}_n (\{ 0\}_n \subseteq \{ -1, 1, 3 \}_m \& x>0)$

$\vee \forall x/\{ 1\}_n (\{ 1\}_n \subseteq \{ -1, 1, 3 \}_m \& x>0)$

.

.

$\vee \forall x/\{ 1, 3 \}_n (\{ 1, 3 \}_n \subseteq \{ -1, 1, 3 \}_m \& x>0)$

.

.

$\vee \forall x/\{ -2, -1, 0, 1, 2, 3 \}_n (\{ -2, -1, 0, 1, 2, 3 \}_n$

$\subseteq \{ -1, 1, 3 \}_m \& x>0) , \sigma, \varphi)$

= T .

Because the formulas $\forall x/\{ 1\}_n (\{ 1\}_n \subseteq \{ -1, 1, 3 \}_m \& x>0)$ and $\forall x/\{ 1, 3 \}_n (\{ 1, 3 \}_n \subseteq \{ -1, 1, 3 \}_m \& x>0)$ are true.

2.3. Some properties of the multi-layer logic

(A) Two theorems on the multi-layer logic are proved. These state that the multi-layer logic defined in Section 2.1 and 2.2 is a proper extension of the many-sorted logic. For proofs, [1] can be referred.

Theorem. 1. Let $G(x_0,\ldots,x_m)$ is a formula and variables $x_{m+1},\ldots,x_n$ are not contained in $G(x_0,\ldots, x_m)$, then

$(Q_n x_n / *_n \ldots *_1 D) \ldots (Q_m x_m / x_{m+1}) \ldots (Q_0 x_0 / x_1) G(x_0, \ldots, x_m)$

$= (Q_m x_m / *_m \ldots *_1 D) \ldots (Q_0 x_0 / x_1) G(x_0, \ldots, x_m)$, where $Q_j = \forall$

$(0 \leq m \leq j \leq n)$.

<u>Corollary. 1.</u>

Putting $m = 0$, we have $(\forall x_n / *_n \ldots *_1 D) \ldots (\forall x_0 / x_1) G(x_0)$

$= (\forall x_0 / D) G(x_0)$.

<u>Lemma. 1.</u>

Let G is a formula and x_n is a free variable in G. If x_n is defined over the domain E and $E \subseteq *_n \ldots *_1 D$, then the formula below is a valid formula.

$(\exists x_n / E)(Q_{n-1} x_{n-1} / x_n) \ldots (Q_0 x_0 / x_1) G(x_0, \ldots, x_n) \rightarrow$

$(\exists y_n / *_n \ldots *_1 D)(Q_{n-1} y_{n-1} / y_n) \ldots (Q_0 y_0 / y_1) G(y_0, \ldots, y_1)$.

<u>Theorem. 2.</u> Let $G(x_0, \ldots, x_m)$ is a formula and variables $x_{m+1}, \ldots, x_n$ are not contained in $G(x_0, \ldots, x_m)$, then

$(Q_n x_n / *_n \ldots *_1 D) \ldots (Q_m x_m / x_{m+1}) \ldots (Q_0 x_0 / x_1) G(x_0, \ldots, x_m)$

$= (Q_m x_m / *_m \ldots *_1 D) \ldots (Q_0 x_0 / x_1) G(x_0, \ldots, x_m)$, where $Q_j = \exists$

$(0 \leq m \leq j \leq n)$.

<u>Corollary. 2.</u>

Putting $m = 0$, we have $(\exists x_n / *_n \ldots *_1 D) \ldots (\exists x_0 / x_1) G(x_0)$

$= (\exists x_0 / D) G(x_0)$.

Corollary 1 together with corollary 2, it is proved that the many-sorted logic is a proper subset of the multi-layer logic defined here. In other words, the former is a special case of the latter.

(B) We provide the implication conditions of literals of the multi-layer logic. They play an essential role in transforming queries. For proofs and examples, [1] can be referred.

Theorem. 3. (Implication conditions of the multi-layer logic)

Let D_x , D_y denote the domains of variables x, y respectively and L(...,x_k) stands for a literal which contains x_k and may contain some other variables. The formula below is valid , if every corresponding variable pair x and y satisfies the set theoretical conditions given Table 1.

$$(Q'_n y_n/*_n \ldots *_1 D_y) \ldots (Q'_k y_k/y_{k+1}) \ldots (Q'_0 y_0/y_1) L(\ldots, y_k)$$
$$\rightarrow (Q_m x_m/*_m \ldots *_1 D_x) \ldots (Q_k x_k/x_{k+1}) \ldots (Q_0 x_0/x_1) L(\ldots, x_k),$$

where min(m, n) $\geq$ k $\geq$ 0.

Q'_k	Q_k	Condition
$\forall$	$\forall$	$D_y \supseteq D_x$
$\forall$	$\exists$	$D_y \wedge D_x \neq \emptyset$
$\exists$	$\exists$	$D_y \subseteq D_x$

Table 1. Implication condition.

(C) We establish the unification rule in the multi-layer logic defined in Section 2.1 and 2.2. Note that Unification Algorithm and Unification Theorem by J.A. Robinson [9] are not directly applic- able to the expressions in the multi-layer logic since the domain of a term is indicated explicitly.

Unification Algorithm in the multi-layer logic

Let A be any finite nonempty set of expressions that satisfy the Theorem 3.

Step 1. Set $k = 0$, $\mu_k = \varepsilon$ and $A_k = A$.

Step 2. If A_k is a singleton, terminate : μ_k is a unifier of A. Otherwise find the disagreement set D_k of A_k.

Step 3. If there are variables x_k, y_k in D_k which are defined over domains X_k, Y_k , then

(1) $\mu_{k+1} = \mu_k\{ x_k/y_k , X_k/Y_k \}$ for $(Q_x, Q_y) = (\forall, \forall)$

(2) $\mu_{k+1} = \mu_k\{ x_k/y_k, \{X_k/ Y_k\}/X_k \}$ for $(Q_x, Q_y) = (\exists ,\forall)$

(3) $\mu_{k+1} = \mu_k\{ y_k/x_k , Y_k/X_k \}$ for $(Q_x, Q_y) = (\exists , \exists)$

where Q_x and Q_y are quantifier of variables x_k and y_k.
If there exist a variable x_k and a term t_k in D_k, then

(4) $\mu_{k+1} = \mu_k\{ t_k/x_k \}$.

Next let $A_{k+1} = A_k\{ r_k/z_k \}$, where $\{ r_k/z_k \}$ is $\{ x_k/y_k , X_k/Y_k \}$, $\{ x_k/y_k , \{X_k/ Y_k\}/X_k \}$, $\{ y_k/x_k , Y_k/X_k \}$ and $\{ t_k/x_k \}$ for (1), (2), (3) and (4) respecively. Go to Step 2.

Otherwise terminate, A is not unifiable.

The unification rule in the multi-layer logic is justified in the following theorem, which corresponds to the Unification Theorem proved by J.A. Robinson [9].

Theorem. 4. (Unification Theorem in the multi-layer logic)
Let A be any finite nonempty set of expressions in the multi-layer logic. If A is unifiable, then the Unification Algorithm terminates with most general unifier μ_A of A.

Proof.

The proof is by induction on k. Let θ be any unifier for A. It suffices to prove that under the hypothesis of the theorem the underlined Unification Algorithm terminates at Step 2, and until the underlined Unification Algorithm terminates , there is some substitution ψ_k such that the equation $\theta = \mu_k\psi_k$ holds at Step 2 ($k \geq 0$).

For $k = 0$, $\theta = \mu_k\psi_k$ holds with $\psi_0 = \theta$ bacause $\mu_0 = \varepsilon$.

Suppose $\theta = \mu_k\psi_k$ holds for $k \geq 0$ at Step 2. If $A\mu_k$ is a singleton, then the Unification Algorithm terminates at Step 2 with $\mu_A = \mu_k$ the most general unifier of A and $\psi = \psi_k$ the required substitution. If $A\mu_k$ is not a singleton, then the underlined Unification Algorithm finds the disagreement set D_k. Since A is unifiable by hypothesis, there must be a variable and perhaps more in D_k. Because, in the multi-layer logic defined here, domain of a term is expressed explicitly, set theoretical relations between domains have to be considered.

Case 1. (D_k contains more than two variables.)

Let x_k, y_k be variables in D_k which are defined over domains X_k, Y_k and preceded by quantifiers Q_x, Q_y. There are three possible cases according to the combination of quantifiers. In the later, let ...($Q_x x_k/X_k$)... G(...,x_k) and ...($Q_y y_k/Y_k$) ... G(...,y_k) be formulas involving x_k and y_k as their arguments. ... denotes they may involve some other variables.

Case 1-a. (Q_x, Q_y) = ($\forall$, $\forall$)

According to Theorem 3

...($\forall\ y_k/Y_k$)... G(...,y_k) $\rightarrow$ ...($\forall\ x_k/X_k$)... G(...,x_k)

is a valid formula iff X_k is a subset of Y_k. However, the converse is not always the case. Because if $Y_k \subset X_k$ then there may exist a element e^ such that $e\hat{} \in X$ and $e\hat{} \notin Y$ and satisfies G(...,e^). Thus, to conserve the validity of formulas concerned, x_k and X_k have to replace y_k and Y_k.

Case 1-b. $(Q_x, Q_y) = (\exists , \forall)$

According to Theorem 3

$\ldots(\forall y_k/Y_k)\ldots G(\ldots,y_k) \rightarrow \ldots(\exists x_k/X_k)\ldots G(\ldots,x_k)$

is a valid formula iff $X_k \wedge Y_k \neq \emptyset$. Since $X_k \wedge Y_k \subseteq X_k$ and $X_k \wedge Y_k \subseteq Y_k$, to conserve the validity of formulas concerned, $X_k \wedge Y_k$ have to replace X_k.

Case 1-c. $(Q_x, Q_y) = (\exists , \exists)$

According to Theorem 3

$\ldots(\exists y_k/Y_k)\ldots G(\ldots,y_k) \rightarrow \ldots(\exists x_k/X_k)\ldots G(\ldots,x_k)$

is a valid formula iff Y_k is a subset of X_k. However, the converse is not always the case. Because if $X_k \subset Y_k$ then there may exist a element e^ such that $e\hat{} \in Y_k$ and $e\hat{} \notin X_k$ and satisfies G(...,e^). Thus, to conserve the validity of formulas concerned, y_k and Y_k have to replace x_k and X_k.

Case 2. (D_k involves only one variable.)

Let x_k, t_k be a variable and a term other than a variable in D_k. Note that x_k can not be occurred in t_k since, by the hypothesis of the theorem, A satisfies the Theorem 3. Hence this case never happen. (For details, [1] can be referred.) In other words, Theorem 3 guarantees that the set A is unifiable.

Thus, in both cases, we can select from D_k a term r_k and a variable z_k that does not occur in r_k. Since ψ_k unifies $A\mu_k$, ψ_k has to unify the disagreement set D_k of $A\mu_k$. Hence z_k and r_k satisfy the equation

$z_k\psi_k = r_k\psi_k$ ----- (A).

Because z_k does not occur in r_k, the underlined Unification Algorithm does not terminate in Step 3, but return to Step 2 with $\mu_{k+1} = \mu_k\{ r_k/z_k \}$.

Let $\psi_{k+1} = \psi_k - \{ z_k\psi_k/z_k \}$. Then we have $r_k\psi_k = r_k\psi_{k+1}$ as r_k does not contain z . Thus we have :

$\mu_k\psi_k = \mu_k\{ z_k\psi_k/z_k \} \cup \psi_{k+1}$ ----- by definition of ψ_{k+1},

$= \mu_k\{ r_k\psi_k/z_k \} \cup \psi_{k+1}$ ----- by equation (A),

$= \mu_k\{ r_k\psi_{k+1}/z_k \} \cup \psi_{k+1}$ ----- z_k does not occur in r_k,

$= \mu_k\{ r_k/z_k \} \cup \psi_{k+1}$ ----- by definition of composition of substitution,

$= \mu_{k+1}\psi_{k+1}$ ----- by definition of μ_{k+1}.

Hence $\theta = \mu_k\psi_k$ holds at Step 2 for all $k \geq 0$ until the underlined Unification Algorithm terminates. This completes the proof.

Table 2 summarizes the unification rule in the multi-layer logic.

Theorem. 5. (Soundness of the query transformation)

If the formulas :

[A1] $(Q'_x, x'/X') [(\exists y/Y) G(x', y)]$,

[A2] $(Q'_x, x'/X') [(\exists y/Y) G(x', y) \rightarrow P(x')]$,

$(Q'_k y_k/Y_k)$	$(Q_k x_k/X_k)$	$(Q''_k z_k/Z_k)$
$(\forall\ y_k/Y_k)$	$(\forall\ x_k/X_k)$	$(\forall\ z_k/X_k)$
$(\forall\ y_k/Y_k)$	$(\exists\ x_k/X_k)$	$(\exists\ z_k/\{X_k \wedge Y_k\})$
$(\exists\ y_k/Y_k)$	$(\exists\ x_k/X_k)$	$(\exists\ z_k/Y_k)$
constant	$(\forall\ x_k/X_k)$	$\emptyset$
constant	$(\exists\ x_k/X_k)$	$\emptyset$
$(\forall\ y_k/Y_k)$	constant	$\emptyset$
$(\exists\ y_k/Y_k)$	constant	$\emptyset$

Table. 2. Unification rule in the multi-layer logic.

[Qu] $(Q_x x/X)(Q_z z/Z)$ [$H(x,z)$ # $P(x)$],

[Ic] $(Q'_{x'} x'/X')\ P(x') \rightarrow (Q_x x/X)\ P(x)$

are true over the interpretation (σ , φ), then the formula

[Tq] $(Q''_{x''} x''/X'')(Q_z z/Z)$[$H(x'',z)$ # $(\exists\ y/Y)\ G(x'' , y)$]

is also true over the interpretation (σ , φ). Where # denotes either & or V, and $Q''_{x''}$, x'' and X'' obey the unification rule given in Table 2.

Proof.

Because [A1] and [A2] are true over the interpretation (σ , φ), $(Q'_{x'} x'/X')\ P(x')$ is true by the definition of $\rightarrow$. Thus $(Q_x x/X)$ $P(x)$ is true, since $(Q'_{x'} x'/X')\ P(x')$ and [Ic] are true over the interpretation (σ , φ). According to Theorem 4, the unification rule given in Table 2 guarantees to find the most general unifier of $(Q'_{x'} x'/X')\ P(x')$ and $(Q_x x/X)\ P(x)$, hence if [A1] and [Ic] are true over the interpretation (σ , φ), then ($Q''_{X''}x''/X''$) G(x', y) is true. Thus [Tq] is true over the interpretation (σ , φ). This completes the proof.

Hypothesis of Theorem 5 are reasonable in the environment of the relational database, which are the subjects of later chapters. For details, [1] can be referred.

3. Describing queries in the multi-layer logic

The application of the multi-layer logic to a relational database query language will be introduced by a series of example queries. The examples of this section are drawn from a database which describes lands. The database contains the following relations :

LYP(LAND, YEAR, PRIC) ;

LU (LAND, USAG) ;

LDA(LAND, DIST, AREA).

The relation LYP has a row which gives the price for each land's identifier and year. The relation LU gives the usage of each land. The relation LDA gives, for each land, its area and the distance from the center of a city in which it is located. In the following formulation, the variable marked by the '^' symbol tells the system to output the value of it. The numerals following the '#' symbol indicate a number. The characters quoted by the symbol "'" denote character constants.

A query against more than one relation with Boolean conditions

Query 1. List the lands of usage a and their areas, which are less than 35 kilometer aprat from the center of a city and whose prices are less than 600,000 YEN/m^2 in the year 1981.

Q1. (∃ l^/LAND)(∃ a^/AREA)(∃ p/PRIC)(∃ d/DIST)

[LYP(l,#1981/YEAR,p) & LU(l,'a'/USAG)

& LDA(l,d,a)

& LT(p,#60/PRIC) & LT(d,#35/DIST)].

The predicate LT(p,#60/PRIC) restricts values of the column PRIC in the relation LYP to less than 60, i.e., the price of a land is less than 600,000 YEN/m^2. Similarly, the predicate LT(d,#35/DIST) restricts values of the column DIST in the relation LDA to less than 35, i.e., the distance of a land from the center of a city is less than 35 kilometer. These predicates play the role of Codd's "restriction" operator.

A query against more than one relation with a universal quantification

Query 2. List the lands which are inquired in all the years, their usage and distance from the center of a city.

Q2. (∃ l^/LAND)(∃ u^/USAG)(∃ d^/DIST)(∀ y/YEAR)

(∃ p/PRIC)(∃ a/AREA)

[LYP(l,y,p) & LU(l,u) & LDA(l,d,a)].

Hereafter, we use an abbreviation in which variables that are not referred in the query description are replaced by "@" symbol. For example, Q2 is abbreviated like Q2'.

Q2'. (∃l^/LAND)(∃ u^/USAG)(∃ d^/DIST)(∀ y/YEAR)

[LYP(l,y,@) & LU(l,u) & LDA(l,d,@)].

A query using built-in arithmetic and aggregation fuctions

Query 3. List, for each land of usage a and inquired in the year 1981, its identifire, its price and the difference of the price with the average price computed on all lands inquired in the year 1981.

<u>Q3.</u> (∃ Q/*PRIC)(∃ l^/LAND)(∃ p^/PRIC)(∃d^/INTG)(∀ q/Q)

```
      [  LYP( l,#1981/YEAR,p ) & LU( l,'a'/USAG )
       & LYP( @,#1981/YEAR,q )
       & LET( d,sub(p,avg(Q)) ) ].
```

Apart from the predicate constant LET(d, sub(p,avg(Q)), this query is decomposed into the following two formulas. One is (∃ l^/LAND)(∃ p^/PRIC) [LYP(l,#1981/YEAR,p) & LU(l,'a'/USAG)]. The result of this formula is a set of values of pairs < l,p > which satisfy the stipulation, that is, a set of lands of usage a and years corresponding to them. The other is (∃ Q/*PRIC)(∀ q/Q) [LYP(@, #1981/YEAR,q)]. This formula defines a multiset Q of prices to which corresponds the multiset of lands inquired in the year 1981. The result of Q4 yields the lands of usage a and the corresponding prices together with the difference of the corresponding price with the average price computed on Q.

A query with a nested aggregation function and a Boolean condition

<u>Query 4.</u> What is the average number of lands per usage, which are inquired in the year 1981 and whose areas are not less than 100 square meter.

<u>Q4.</u> (∃ LL/**LAND)(∃ g^/REAL)(∀ u/USAG)(∃ L/LL)(∀ l/L)(∃ a/AREA)

```
      [  LU( l,u ) & LYP( l,#1981/YEAR,@ ) & LDA( l,@,a )
       & GE( a,#100/AREA )
```

& LET(g,avg(ucount(LL)))].

In the query 4, it is required to calculate "an average number of lands". For this purpose, we have to introduce a variable LL whose value is a powerset of lands. This fact is explicitly represented by a prefix ($\exists$ LL/**LAND). By means of a variable LL, the underlined specification is easily expressed by a nested aggregation function, i.e., avg(ucount(LL)). For example, if the value of a variable LL is a powerset { $\{A1, A2, A3\}_a$, $\{A3, B2\}_b$, $\{D2\}_c$ }, where $\{\ \}_x$ indicates a multiset corresponding to the value x. Then, ucount(LL) yields a set {3, 2, 1 }. Consequently, the function avg(ucount(LL)) yields 2.

4. Virtual relation and inference algorithm

In the multi-layer logic for the relational database, virtual relations are defined through formulas which have the following form :

$$(\forall x_1/X_1) \ldots (\forall x_p/X_p)\ [\ (\exists x_{p+1}/X_{p+1}) \ldots (\exists x_n/X_n)$$
$$(\ R_1 \# \ldots \# R_m \# S\ ) \rightarrow T(\ x_{i_1}, \ldots, x_{i_q}\ )\ ]\ ,$$

where # is either & or V, each of $R_1, \ldots, R_m$ is a formula containing base or virtual relations connected by either & or V. T ($x_{i_1}, \ldots, x_{i_q}$), ($i_1, \ldots, i_q \in \{ 1, \ldots, p \}$), is an atomic formula containing only variables $x_{i_1}, \ldots, x_{i_q}$ and denotes a virtual relation to be defined. S is a formula free of both base and virtual relations. S may contain predicate constants such as EQ, LT, GE, LET, etc.

A query including virtual relations could not be reduced directly to the retrieval procedures or a sequence of operations in the relational algebra. To evaluate such a query, it is necessary to transform a query into one that contains no virtual relation.

The method for transforming a query is based on the idea of replacing virtual relations (if any) in the query by those defining formulas. The substitutions are done repeatedly until a given query contains no virtual relation.. Now, an overview of the inference algorithm is given. Generally, the query containing virtual relations is given as :

(1) --- ... $(Q_1 x_1/X_1) \ldots (Q_p x_p/X_p) \ldots$ [...$T(x_{i_1},\ldots,x_{i_q})$...],

where $T(x_{i_1},\ldots,x_{i_q})$ is a q-place virtual relation, $i_1,\ldots,i_q \in \{1,\ldots,p\}$, Q's are quantifiers of either $\forall$ or $\exists$, and ... indicates that the query may contain other variables, constants or functions.

The inference algorithm consists of the following steps.

Step 1. If the query (1) contains predicate constants which can be evaluated at this point, then evaluate them. Otherwise, simply go to next step.

Step 2. Find a virtual relation in a query. If there is, go to next step, otherwise go to "reduction procedure" which is discussed in section 5.

Step 3. Let a formula defining a virtual relation $T(y_{i_1},\ldots,y_{i_q})$ be given as :

(2) ---

$$(\forall y_1/Y_1) \ldots (\forall y_p/Y_p) \, [\, (\exists y_{p+1}/Y_{p+1}) \ldots (\exists y_n/Y_n) \, [\, (R_1 \# \ldots \# R_m \# S) \rightarrow T(y_{i_1},\ldots,y_{i_q}) \,].$$

Rename the variable names so that a query and a formula (2) share no variables in common.

Step 4. If a formula $(\forall y_1/Y_1)\ldots(\forall y_p/Y_p)\ T(y_{i_1},\ldots,y_{i_q}) \rightarrow$

$$(Q_1 x_1/X_1)\dots(Q_p x_p/X_p)\ T(x_{i_1},\dots,x_{i_q})$$

is a valid formula, then substitute $T(x_{i_1},\dots,x_{i_q})$ by $(\forall\ y_1/Y_1)\dots(\forall\ y_p/Y_p)\ [\ (\exists\ y_{p+1}/Y_{p+1})\ \dots\ (\exists\ y_n/Y_n)\ [\ R_1 \&\ \dots\ \&\ R_m\ \&\ S\ ]$. As the result, the query (1) is transformed to the formula (3). The validity of the formula

$$(Q_1' y_1/Y_1)\dots(Q_p' y_p/Y_p)\ T(y_{i_1},\dots,y_{i_q}) \rightarrow (Q_1 x_1/X_1)\dots(Q_p x_p/X_p)\ T(x_{i_1},\dots,x_{i_q})$$

is testable by the "implication conditions" shown in Table 1.

(3)---

$$\dots\ (Q_1^{\sim} z_1/Z_1)\dots(Q_p^{\sim} z_p/Z_p)\dots(\exists\ y_{p+1}/Y_{p+1})\ \dots\ (\exists\ y_n/Y_n)$$
$$[\ \dots\ R_1^{\sim}\ \#\dots\#\ R_m^{\sim}\ \#\ S^{\sim}\dots\ ],$$

where $Q_j^{\sim}$ $,z_j,Z_j$ $(1<j<p)$, $R_1^{\sim},\dots,R_m^{\sim},S^{\sim}$ are respectively obtained from $Q_j,R_1,\dots,R_m,S$ by unification rules shown in Table 2.

If a formula $(\forall\ y_1/Y_1)\dots(\forall\ y_p/Y_p)\ T(y_{i_1},\dots,y_{i_q}) \rightarrow (Q_1 x_1/X_1)\dots(Q_p x_p/X_p)\ T(x_{i_1},\dots,x_{i_q})$ is not a valid formula, then try for other formulas defining $T(y_{i_1},\dots,y_{i_q})$.

The general flow-chart of the inference algorithm for the SBDS-F3 is shown in Figure 1.

5. Reducing a formula into database access procedure

The flowchart of the reduction algorithm is shown in Figure 2. The algorithm which reduces a formula in the first order logic into a sequence of the operations on the relational algebra is discussed by E.F.Codd[5], R.Reiter[8] and C.L.Chang[6]. But their algorithm is not applicable to a

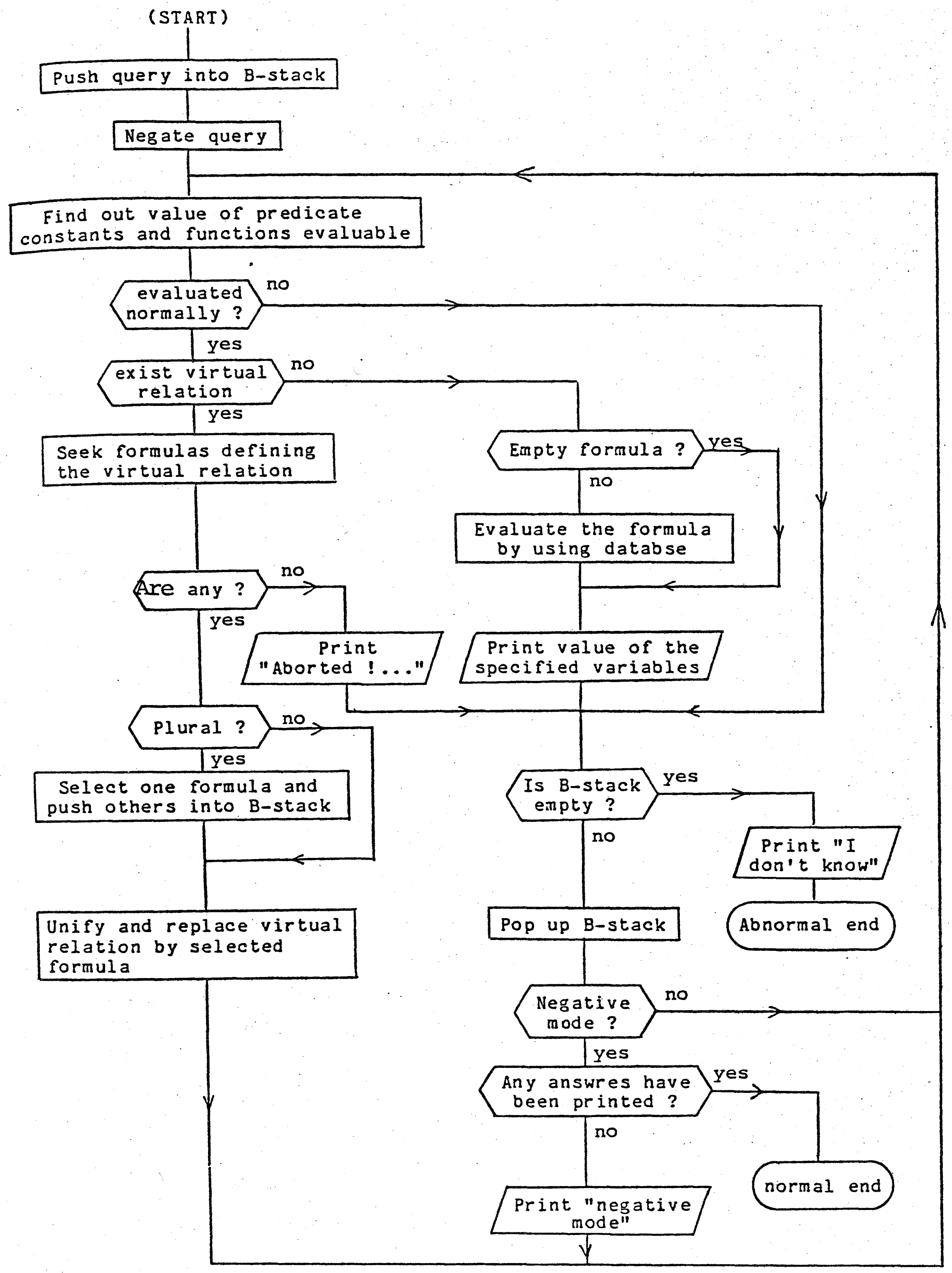
(START)
Push query into B-stack
Negate query
Find out value of predicate constants and functions evaluable
evaluated normally ?
no
yes
exist virtual relation
no
yes
Seek formulas defining the virtual relation
Empty formula ?
yes
no
Evaluate the formula by using databse
Are any ?
no
yes
Print "Aborted !..."
Print value of the specified variables
Plural ?
no
yes
Select one formula and push others into B-stack
Is B-stack empty ?
yes
no
Print "I don't know"
Unify and replace virtual relation by selected formula
Pop up B-stack
Abnormal end
Negative mode ?
no
yes
Any answres have been printed ?
yes
no
normal end
Print "negative mode"

Figure 1. The general flow-chart of the inference algorithm for the SBDS-F3.

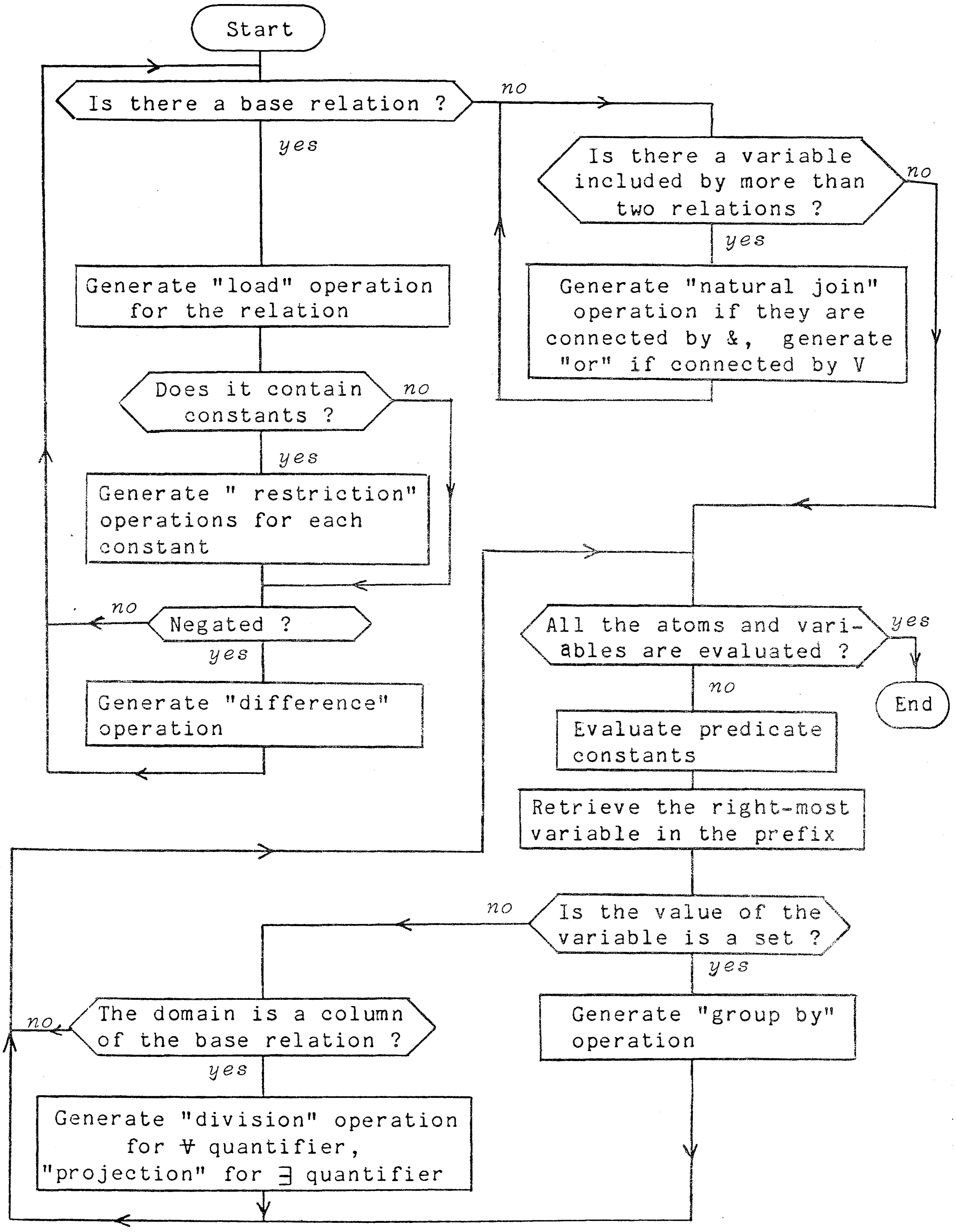
Start
Is there a base relation ?
no
yes
Generate "load" operation for the relation
Does it contain constants ?
no
yes
Generate " restriction" operations for each constant
no
Negated ?
yes
Generate "difference" operation
Is there a variable included by more than two relations ?
no
yes
Generate "natural join" operation if they are connected by &, generate "or" if connected by V
All the atoms and variables are evaluated ?
yes
End
no
Evaluate predicate constants
Retrieve the right-most variable in the prefix
no
Is the value of the variable is a set ?
yes
Generate "group by" operation
no
The domain is a column of the base relation ?
yes
Generate "division" operation for ∀ quantifier, "projection" for ∃ quantifier

Figure 2. The flowchart of the reduction algorithm.

formula in the multi-layer logic, because it may contain variables whose values are sets. The reduction algorithm for the first order predicate logic is extended to manipulate the formula in the multi-layer logic by containing the "determine a set" operation explicitly.

Overview of the algorithm is as follows. First, an atom corresponding to a base relation is translated into the "load" operation, an operation to access the base relation. If the atom contain any constants and/or predicate constants evaluable, they are translated into the "restriction" operation. This process is repeated until all the atoms corresponding to base relations are translated. Next, all the variables included by more than two relations are translated into the "natural join" operation if the relations are connected by &, the "or" operation if the relations are connected by V. (Note that "intersection" operation is a special case of the "natural join" operation.) If all the atoms in the formula are evaluated , then the reduction process stops. Otherwise predicate constants are evaluated and quantifiers of variables are translated.

Translation of quantifiers consists of the following process. First retrieve the right-most variable in the prefix. If its value is a set, then it is translated into the "determine a set" operation. Otherwise, if the domain of the variable is a column of a base relation, then the ∀ quantifier is translated into the "division" operation, the ∃ quantifier is into the "projection" operation.

The notation of the operations generated from a formula are summarized below.

```
*LOAD <relation name> [ <work file name>
                        <column identifier list> ]
*OR   [ <work file name> ]
         [ <work file name><column identifier list> ],
         [ <work file name><column identifier list> ]
*AND  [ <work file name> ]
         [ <work file name><column identifier list> ],
         [ <work file name><column identifier list> ]
*DIFF [ <work file name> ]
         [ <work file name><column identifier list> ],
         [ <work file name><column identifier list> ]
*PROJ  [ <work file name><column identifier list> ]
                       $ <column identifier>
*JOIN  [ <work file name> ]
       [ <work file name><column identifier list> ]
    < <column identifier list> >< <column identifier list> >
       [ <work file name><column identifier list> ]
*REST  <work file name> : <column identifier>
      <predicate constant> <constant>/<domain specification>
*DIV  [ <work file name><column identifier list> ]
                      / <column identifier>
*DETM  [ <set name> ] = <variable>
       = { <work file name> : <column identifier> FOR EACH
           <work file name> : <column identifier list> }
*DETM  [ <set name> ] = <variable> = { [ <set name> ] }
```

*LOAD means load the base relation indicated. *OR ,*AND and *DIFF denote OR, AND and subtract operations, respectively. *PROJ, *JOIN, *REST and *DIV stand for projection, join,

```
SBDS-F3      Date & Time 12/ 5/81/ 10:43
file  name  = DF04
NO.of lines = 50

        C          ===   Start of database defininfion    ===
        C          ===   Definition of the skeleton structure    ===
    1]  SR         ;LAND,C
    2]  SD         ;LA_SALE,LAND    ,SALE
    3]  SD         ;LA_N_SA,LAND    ,SALE
    4]  SD         ;LA_PRIV,LAND    ,POSS
    5]  SD         ;LA_NAT ,LAND    ,POSS
    6]  SI         ;LA_P_S ,LA_PRIV,LA_SALE
    7]  SI         ;LA_N_S ,LA_NAT ,LA_SALE
    8]  SS         ;SLA_P_S,LA_P_S
    9]  SR         ;YEAR,I
   10]  SR         ;PRIC,I
   11]  SR         ;USAG,C
   12]  SR         ;DIST,I
   13]  SR         ;AREA,I

        C    ===    The LYP relation has a row for each land'
        C       identifier and year, giving its pric.    ===
   14]  RD         ; (LYP, LAND, YEAR, PRIC )
    1)    A1,         #1977,     #50
    2)    A1,         #1979,     #55
    3)    A1,         #1981,     #58
    4)    A2,         #1979,     #80
    5)    A2,         #1981,     #86
    6)    A3,         #1979,     #105
    7)    A3,         #1981,     #112
    8)    B1,         #1SF9,     #63
    9)    C1,         #1977,     #22
   10)    C1,         #1979,     #25
   11)    C2,         #1977,     #23
   12)    C2,         #1979,     #26
   13)    C2,         #1981,     #27
   14)    D1,         #1981,     #40
   15)    D2,         #1981,     #36
   16)    D3,         #1979,     #18
   17)    D3,         #1981,     #20
   18)    %

        C    ===    The LU relation gives the usage of each land.    ===
   15]  RD         ; ( LU, LAND, USAG )
    1)    A1,         a
    2)    A2,         a
    3)    A3,         a
    4)    A3,         b
    5)    B1,         b
    6)    B1,         c
    7)    C1,         c
    8)    C2,         a
    9)    C2,         c
   10)    D1,         d
   11)    D2,         b
   12)    D2,         d
   13)    D3,         d
   14)    %
```

Figure 3.   Some results of the database system SBDS-F3.

```
     C    ===   The LDA relation gives, for each land, its area and
     C    distance from the center of a city in which it is located.
16]  RD        ; ( LDA, LAND, DIST, AREA )
 1)    A1,        #40,        #300
 2)    A2,        #25,        #120
 3)    A3,        #7,         #110
 4)    B1,        #27,        #60
 5)    C1,        #10,        #50
 6)    C2,        #34,        #90
 7)    D1,        #20,        #55
 8)    D2,        #30,        #140
 9)    D3,        #28,        #80
'0)   %
     C        ===    End ofbase definitn    ===



     C    ===   Definition of virtual relations.    ===

     C    ===   The easy-to buy lands (EB_LAND) are the lands for
     C       sale whose prices are less than  250,000 YEN/m .    ===
 7]  KD        ; ( A  l/LA_SALE, A y/YEAR, A p/PRIC ) '
          [ -( (LYP, l, y, p) & (LT, p, #25/PRIC) ) '
          V (EB_LAND, l, y, p) ]

     C    ===   The CU_LAND lands are lands of usage 'c' which
     C       are possessed privately.    ===
  ]  KD        ; ( A l/LA_PRIV ) [ -(LU, l, c/USAG) V (CU_LAND, l) ]

     C    ===   The DN_LAND lands are national lands of usage 'd'.
     KD        ; ( A l/LA_NAT  ) [ -(LU, l, d/USAG) V (DN_LAND, l) ]

     C    ===   The TOWN_L lands are lands which are less than 7 Km
     C       apart from the center of a city.    ===
     KD        ; ( A l/LAND, A d/DIST ) '
          [ -( (LDA, l,d,@) & (LT, d, #7/DIST) ) '
           V (TOWN_L, l, d) ]

          == The LAND81 lands are lands inquired in the year 1981.
               ; ( A l/LAND, A p/PRIC, A u/USAG, A a/AREA, A d/DIST ) '
          [ -( (LYP, l, #1981/YEAR, p) & (LU, l,u) '
                (LDA, l, d, a) ) V (LAND81, l, p, u, a, d) ]

          ===   The Large land81 (LA_LA81) lands are lands inquired
               the year 1981 and whose area is not less than 100 squar
               meter.    ===
               ; ( A l/LAND, A p/PRIC, A u/USAG, A a/AREA, A d/DIST ) '
             [ -( (LU, l, u) & (LYP, l, #1981/YEAR, p) '
                & (LDA, l, d, a) & -(LT, a, #100/AREA) )'
             V (LA_LA81, l, p, u) ]
```

(Continued)

```
        C    ===    Query 1  in Chapter 2.    ===
    23]  QU        ; ( E la^/LAND, E ar^/AREA, E pr/PRIC, E di/DIST ) '
             [ (LAND81, la, pr, a/USAG, ar, di) '
              &(LT, pr, #60/PRIC) & (LT, di, #35/DIST) ] ?
*LOAD   LYP  [ W 1,  1,  2,  3 ]
*REST   W 1:  2 = #1981/YEAR
*REST   W 1:  3 .LT. #60/PRIC
*LOAD   LU  [ W 2,  1,  2 ]
*REST   W 2:  2 = a/USAG
*LOAD   LDA  [ W 3,  1,  2,  3 ]
*REST   W 3:  2 .LT. #35/DIST
*JOIN   [:W 1 ]  [ W 1,  1,  2,  3 ]
        <  1 ><  1 >  [ W 2,  1,  2 ]
*JOIN   [:W 1 ]  [ W 1,  1,  2,  3,  4 ]
        <  1 ><  1 >  [ W 3,  1,  2,  3 ]
*PROJ   [ W 1,  1,  2,  3,  4,  5,  6 ] $  5
*PROJ   [ W 1,  1,  2,  3,  4,  6 ] $  3
*PROJ   [ W 1,  1,  2,  4,V  6 ] $  6
*PROJ   [ W 1,V  1,  2,  4,V  6 ] $  1

== ANSWER == ( la  ar )
 C2            90


        C    ===    Query 4   in Chapter 2.    ===
    24]  QU        ; ( E LL/**LAND, E g^/REAL, A u/USAG, '
                       E L/LL, A 1/L ) '
                   [ (LA_LA81, 1, @, u) '
                   & ( LET, g, avg(ucount(LL)) ) ] ?
*LOAD   LU  [ W 1,  1,  2 ]
*LOAD   LYP  [ W 2,  1,  2,  3 ]
*REST   W 2:  2 = #1981/YEAR
*LOAD   LDA  [ W 3,  1,  2,  3 ]
*REST   W 3:  3 .-LT. #100/AREA
*JOIN   [ W 1 ]  [ W 1,  1,  2 ]
        <  1 ><  1 >  [ W 2,  1,  2,  3 ]
*JOIN   [ W 1 ]  [ W 1,  1,  2,  3,  4 ]
        <  1 ><  1 >  [ W 3,  1,  2,  3 ]
*PROJ   [ W 1,  1,  2,  3,  4,  5,  6 ] $  5
*PROJ   [ W 1,  1,  2,  3,  4,  6 ] $  4
*PROJ   [ W 1,  1,  2,  3,  6 ] $  6
*DETM   [ S 1 ] = L = { W 1:  1  FOR EACH  W 1:  2 }
*DIV    [ W 1,  1,  2,  3 ] /  1
*DIV    [ W 1,  2,  3 ] /  2
*DETM   [ S 2 ] = LL = { [ S 1 ] }
*EVAL   g <= avg

== ANSWER == ( g )
   0.200E  1
```

Figure 3.   (Continued)

restriction and division operations, respectively. *DETM indicates "determine a set" operation, which determines the value of the variable of level m ($m \geq 1$) and stores it in a set indicated. Some results of application of the reduction algorithm are given in Figure 3.

ACKNOWLEDGEMENTS

I wish acknowledge all the members of Meetings of Information Systems for their encouragements and constructive comments.

I am grateful to Dr.N.Yonezaki, Tokyo Institute of Technology, for many helpful discussions and providing me valuable documents.

I would like to express my appreciation to Dr.K.Agusa, Ohno Lab. of Kyoto University , for providing the SAFE editor system. The SAFE system is very useful and exclusively used for preparing this paper.

REFERENCES

1] Udagawa,Y " A Study on Design and Implementation of a Database System Based on Predicate Logic," Doctorial Thesis, Tokyo University, February,1982.

2] Udagawa,Y. and Ohsuga,S. "Design and implementation of a database system based on the multi-layer logic," Proc. of Advanced Database Symposium (Dec. 1981) pp31-42.

3] Udagawa,Y. and Ohsuga,S. "Construction of the multi-layer logic for a relational database query language," Proc. 23th annual conf. of Information Processing (1981) pp447-448.

4] Udagawa,Y. and Ohsuga,S. "GOING --- A Data Sublanguage Using a Graphics Display," WGDB Meeting of IPSJ 29-3 (March, 1982).

5] Codd,E.F. "Relational completeness of data base sublanguages," Data Base Systems , Courant Computer Science Symposia Series , Vol.6 (R. Rustin, Ed.) , Prentice-Hall, (1971),pp65-98

6] Chang,C.L. "DEDUCE 2 : Futher Investigation on deduction in relational data base ," In Logic and Data Bases (H.Gallaire etc.,ed.), Plenum Press,N.Y.,pp201-236(1978).

7] Kunifuji,S. "Second-order many-sorted Boolean logic for knowledge representation language," IIas of Fujitsu reseach report No.14 (Febr. 1981).

8] Reiter,R. "On closed world data bases," Logic and Data Bases (H. Gallaire and J. Minker, Eds.), Plenum Press, New York, (1978), pp55-76.

9] Robinson,J.A. "A machine oriented logic based on the resolution principle," JACM (Jan. 1965), pp25-41.