数理解析研究所講究録
第 461 巻 1982 年 21-48

IMPLEMENTATION OF GOING : A DATALANGUAGE USING GRAPHICS DISPLAY

Yosihisa Udagawa
and Setsuo Ohsuga

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

1. Introduction

This paper concerns design and implementation of a friendly user interface language for the relational databases. A less procedural or non-procedural language is considered to be preferable to end user languages, bacause it permits a user to focus on the functional specifications of a given problem rather than on its solution methodology.

So far, a number of non-procedural interface languages have been proposed for expressing queries against relational databases. Among them, query languages based on the predicate logic 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 the query ;

However, it is pointed out that this class of languages

(1) require users to invent variables to formulate a query ;

(2) rely on the universal and existential quantifiers in the formulation of queries.

Two approaches are commonly known to avoid the difficulty of the use of variables and quantifiers in the predicate logic. One is to describe a query by means of (restricted) natural language. Then it is translated into a formal query, and if necessary, it is rephrased back into a natural language for user's approval. However, this approach meets only limited queries.

The other is to express a query by means of a graph. By taking advantage of two-dimensional representation of a query by using a graphics display , queries are expressed within a simple and easy-to-understand conceptual framework.

GOING (a Graphics Oriented INteractive query lanGuage) , the language discussed in this paper, belongs to the latter approach. GOING is designed to enable the user to express queries in terms of nodes, arcs, comparison predicates and functions. Other well known language with a similar basic orientation includes Query-by-Example [7] and CUPID [6].

The main features of GOING are as follows :

(1) GOING provides a concise , easy-to-understand representation of queries. Queries are expressed in terms of simple figures (ellipse for domains, directed arcs for logical orders of entities), and expression composed of comparison predicates and functions.

(2) GOING avoids the use of quantifiers and bound variables in

expressing queries, hence does not require the user to have a high degree of sophistication on the predicate logic.

(3) The user can control the size and layout of a graph, thus can express a wide variety of queries on a graphics display.

(4) GOING enables the user to state a query in a non-procedural way. The meaning of a query only depends on the properties of the figure expressing it , and does not depend on the sequence of making it. In this sence, a GOING query expression is descriptive.

(5) GOING is designed to minimize the number of concepts that the user subsequently has to learn in order to use the whole language.

(6) A query expression in GOING is translated into an intermediate language, the multi-layer logic in this case, then reduced into a sequence of high level procedural operations of the relational algebra.

(7) GOING provides high expressive power. GOING is able to describe any formula in the multi-layer logic for relational databases [1], which is proved to have more expressive power than conventional query languages based on the predicate logic and graphics-oriented query languages [1,2,3,4].

In section 2, we illustrate GOING and compare it with two other graphics oriented languages, i.e. CUPID, Query-by-Example. Section 3 discusses correspondence between GOING expression and basic concepts in the multi-layer logic. Section 4 deals with implementation of GOING.

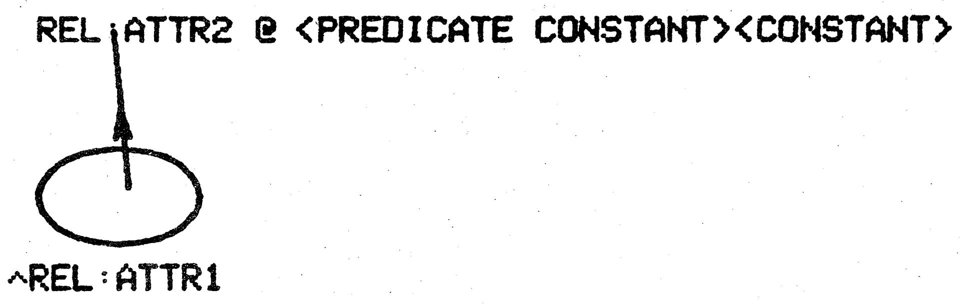
2. GOING expressions for relational database queries

2.1. GOING expressions for the basic database queries

In this section , basic queries of the relational database are illustrated in the GOING expressions.

(A) To express queries containing Boolean conditions.

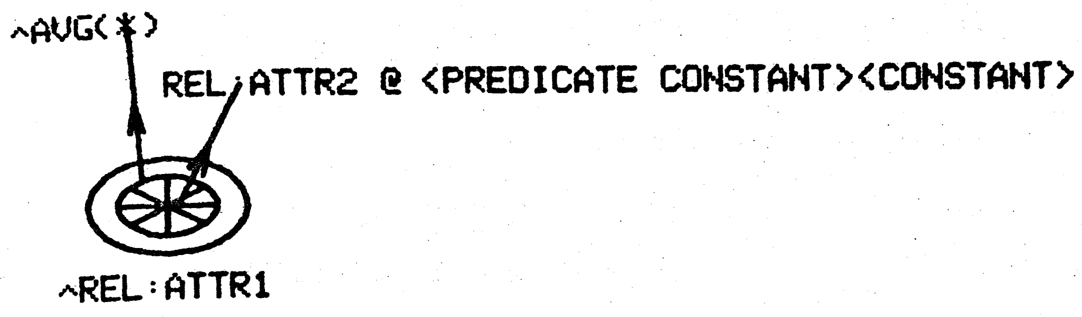
One of the most basic and important queries are those of retrieving values which satisfy given Boolean conditions. In GOING , those queries are expressed in terms of domain specifications, Boolean expressions and directed arcs. For example,



indicates to retrieve the values of ATTR1 in the relation REL relation whose associating values of ATTR2 satisfy the given condition.

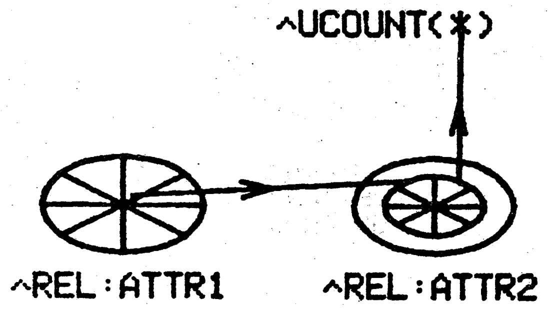
(B) To express queries containing aggregation functions.

Many of practical queries contain aggregation functions. To express these queries, sets to which aggregation functions are applied have to be specified. Because the query language GOING deals with a subset of a given domain explicitly, these queries are formulated in a simple manner. For example, to get the average value of the elements that satisfy a given condition is expressed by the following GOING expression.



(C) To express queries containing "group by" operations.

In some cases, queries involve a aggregation functions whose argument sets are determined for each element of some other set (i.e. for all elements of some set, there exists a subset of a set which is an argument of a given aggregation function). This kind of queries are formulated in a simple manner. For example, the query getting the unique number of items of REL:ATTR2 that are determined for each element of a set REL:ATTR1 is expressed by the following GOING expression.



2.2. Describing queries in terms of GOING expressions, comparing GOING with CUPID and Query-by-Example

In this section , the query language GOING is illustrated by means of example queries and is compared with other graprhics oriented languages, i.e. CUPID, Query-by-Example. The database consists of the following relations:

LYP(LAND, YEAR, PRIC) ;

LU (LAND, USAG) ;

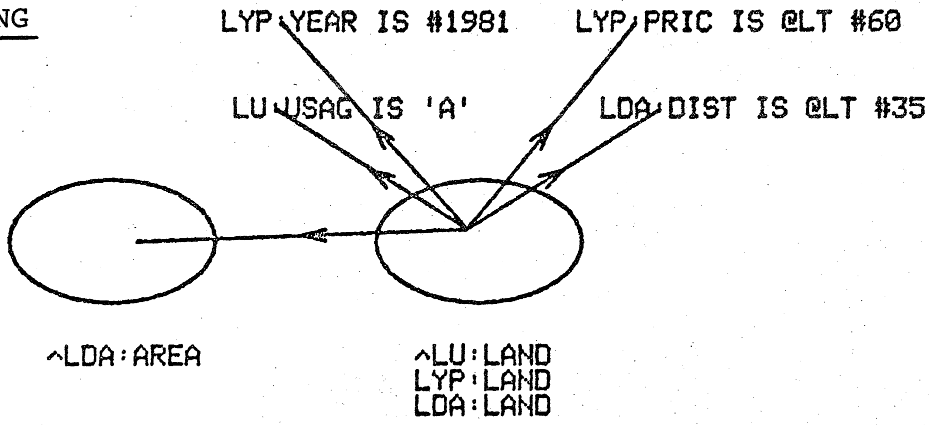
LDA(LAND, DIST, AREA).

The relation LYP has a row giving 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.

A query against more than one relation with Boolean conditions.

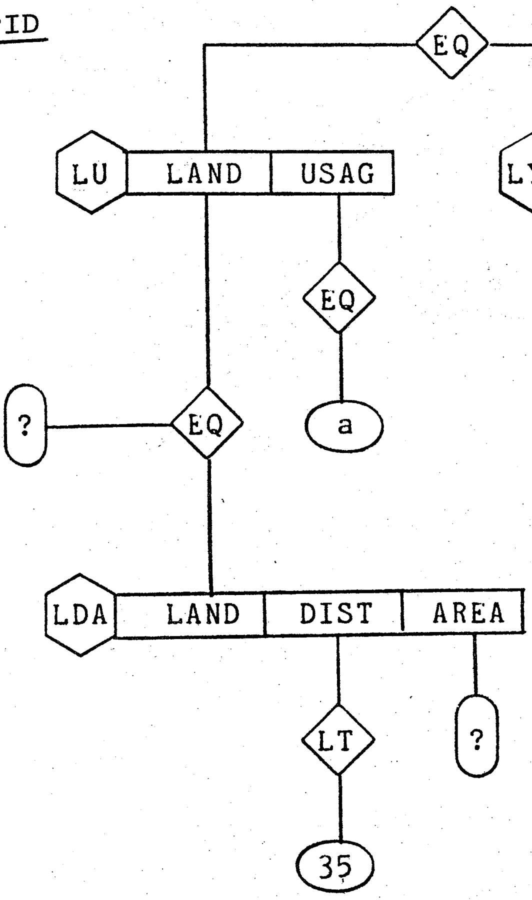
Query 1. List the lands of usage a and their areas, which are less than 35 kilometer aprat from the center of a city and whose prices are less than 600,000 YEN/m^2 in the year 1981.

GOING



A domain or literal expression preceded by "^" symbol denotes to list the value of it. The arc which connects inside of the right ellipse and the first argument of the Boolean predicate LU:USAG IS 'A' indicates that there are some lands in the column LAND in the relations LU, LYP, LDA whose usages are 'A'. The literal expressions LYP:PRIC IS @LT #60 and LDA:DIST IS @LT #35 denote that the values of the price in the relation LYP and the values of the distance in the relation LDA are less than 60 and 35. They are correspond to the predicate constants (∃ p/PRIC) LT(p, #60) and (∃ d/DIST) LT(d, #35/DIST), respectively.

CUPID



Query-by-Example

LU	LAND	USAG
	P.1	'a'

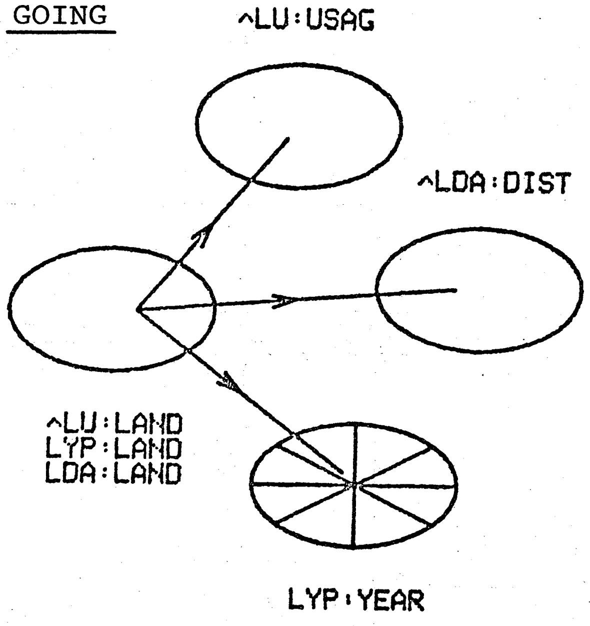
LDA	LAND	DIST	AREA
	1	<'35'	P.

LYP	LAND	YEAR
	1	'1981'

A query against more than one relation with a universal quantification.

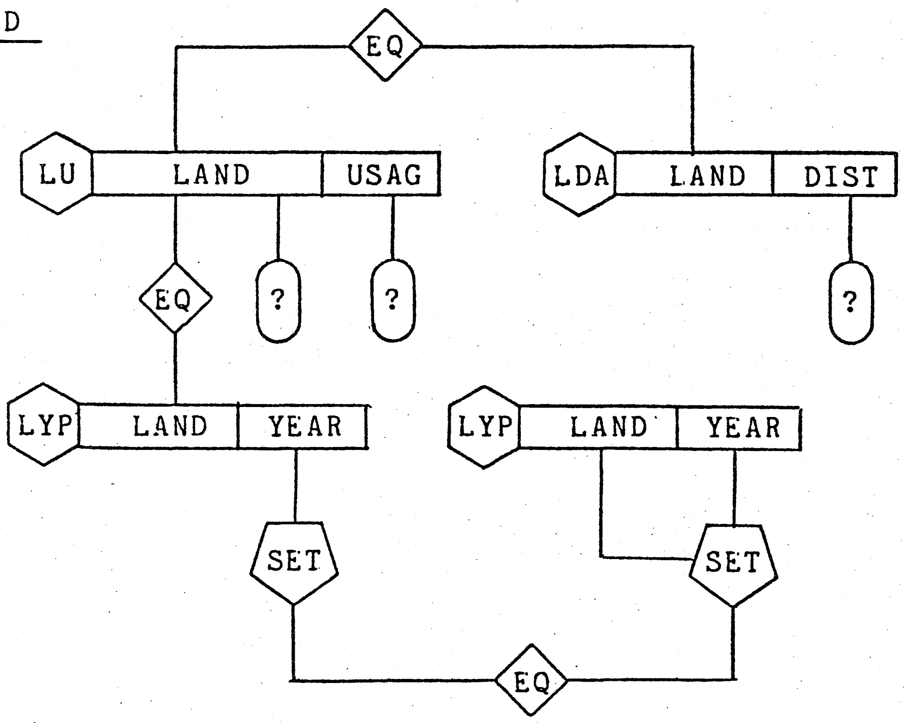
Query 2. List the lands which are inquired in all the years, their usage and distance from the center of a city.

GOING



Note that only those domains referred need be represented in the GOING expression. In this example, the column PRIC in the relation LYP and column AREA in the relation LDA are not used in the above expression.

CUPID



Query-by-Example

LYP	LAND	YEAR
	P. 1	all

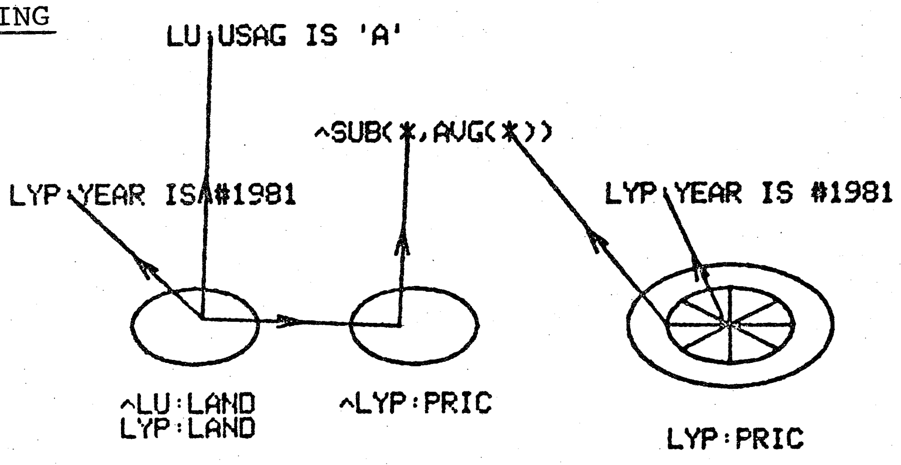
LU	LAND	USAG
	1	P.

LDA	LAND	DIST
	1	P.

A query using built-in arithmetic and aggregation fuctions.

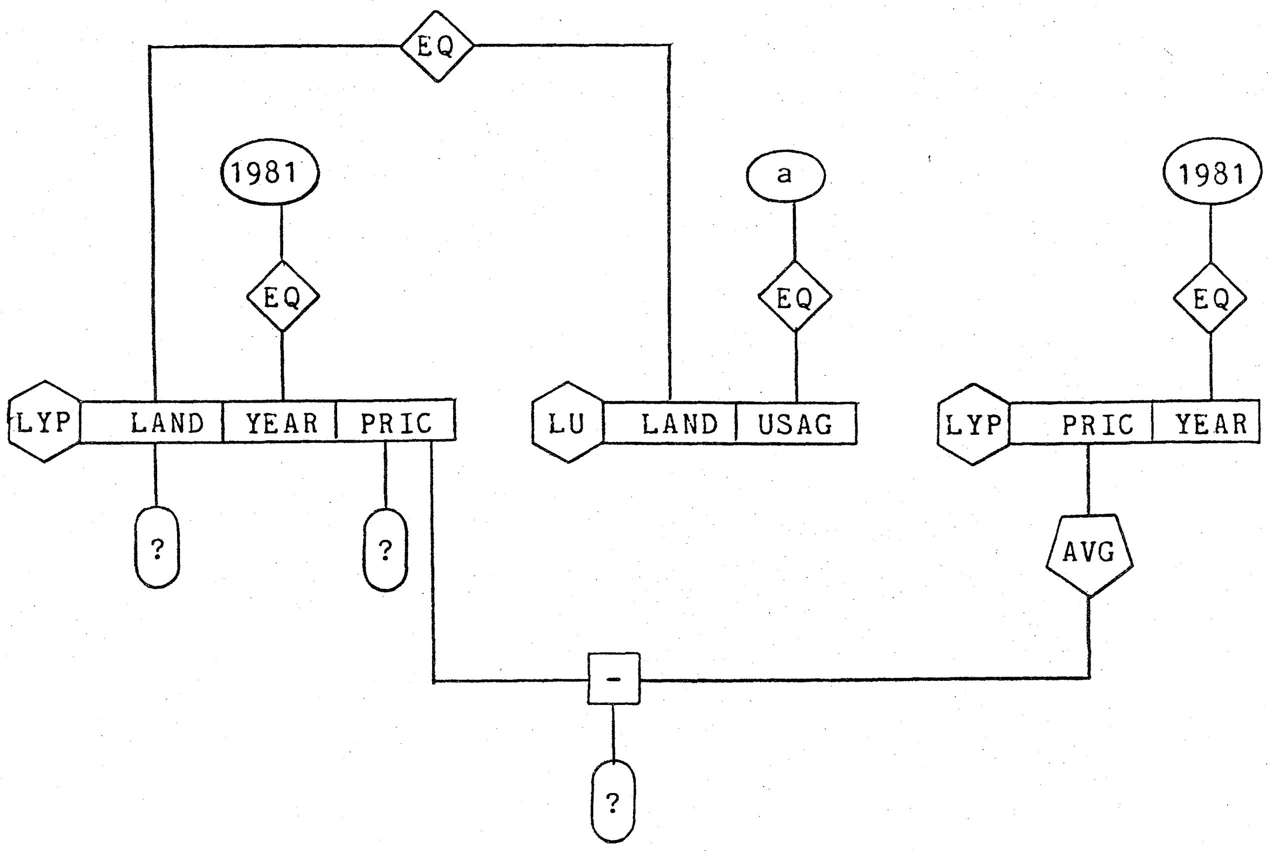
Query 3. List, for each land of usage a and inquired in the year 1981, its identifire, its price and the difference of the price with the average price computed on all lands inquired in the year 1981.

GOING



The literal expression SUB(*,AVG(*)) demonstrates that a built-in function may be nested to any levels so far as the value of arguments are properly defined. Note that first argument of SUB(*, AVG(*)) is some values of column PRIC in the relation LYP, while the argument of AVG(*) is some subsets of column PRIC. This difference is explicitly expressed in the above GOING expression.

CUPID



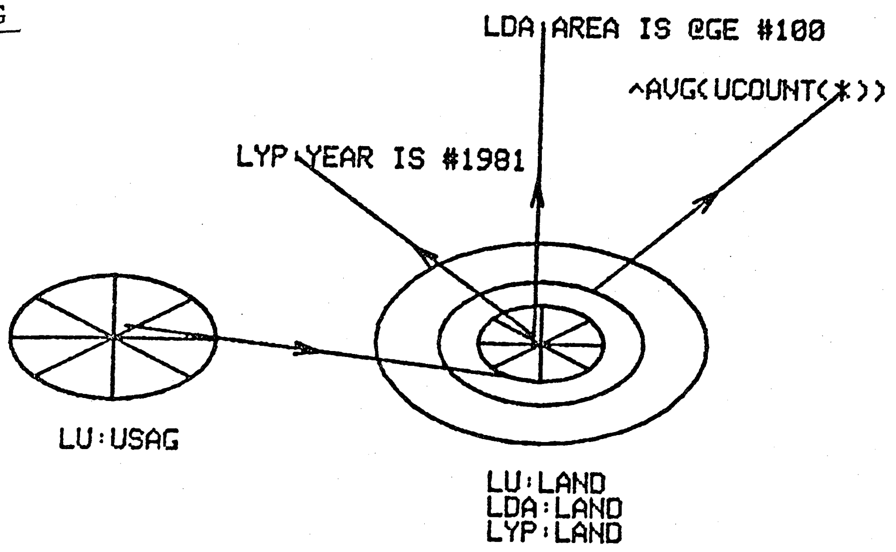
Query-by-Example :

Not expressible.

A query with a nested aggregation function and a Boolean condition.

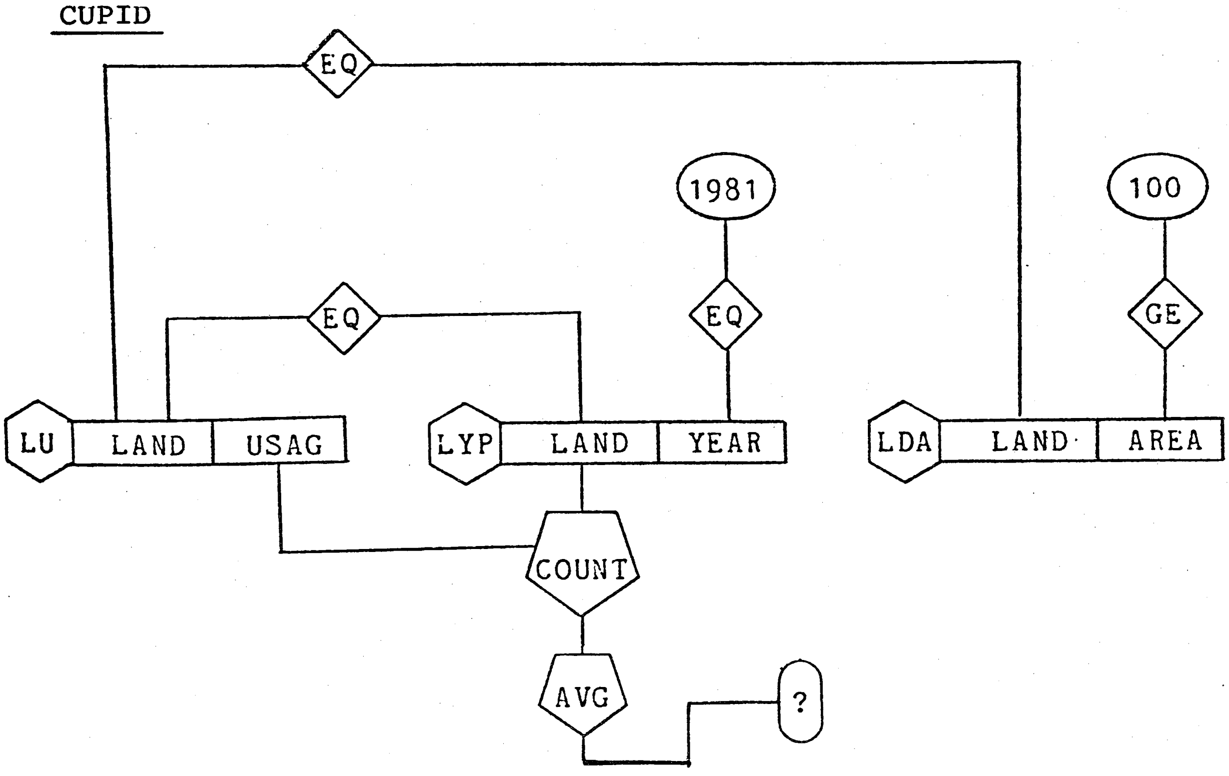
Query 4. What is the average number of lands per usage, which are inquired in the year 1981 and whose areas are not less than 100 square meter.

GOING



This query contains the nested aggregation function AVG(UCOUNT(*)). The argument of this function is a set of subsets of lands inquired in the year 1981 and corresponding area is greater than or equal to (not less than) 100 square meter. The subsets of lands are determined for each usage. This fact is represented by the arc connecting inside of the ellipse LU:USAG and the innermost ellipse of the right figure.

CUPID



Query-by-Example: Not expressible.

3. Correspondence between GOING expression and basic concepts in the multi-layer logic

In this section correspondence between GOING expression and basic concepts in the multi-layer logic is discussed.

As discussed in [1,5], a formula expressing query in the multi-layer logic is constructed in terms of ;

(M-1) relationships between a variable and its domain,

(M-2) quantification of variables,

(M-3) expressions involving predicate constants and/or functions,

(M-4) logical order of quantifiers.

(M-5) logical connection of atomic (literal) formulas, i.e. in terms of & (AND) and V (OR).

GOING represents these basic concepts in the multi-layer logic by means of ;

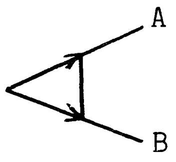
(G-1) ellipse with a domain name for a domain of a variable in a formula,

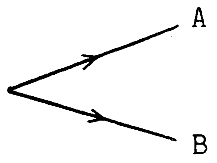
(G-2) hatching on an ellipse for a universal quantification and non-hatching for an existential quantification,

(G-3) Boolean expressions and/or functional expressions having the syntax defined in the appendix,

(G-4) directed arcs mentioned below,

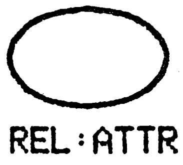
(G-5) undirected arcs that connect specified directed arcs, respectively.

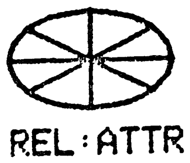
Directed arc in (G-4) is used for specifying logical order of quantifiers , that is, the arc x $\longrightarrow$ y denotes that x determines the corresponding entity y. An undirected arc 

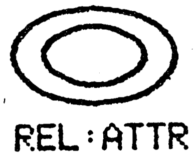
indicates that two Boolean conditions A and B are ORed. If there is no specification, it represents ANDed. That is, 

indicates that the Boolean condition A and B are ANDed.

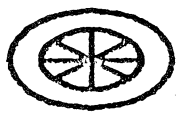
The GOING expressions "REL:ATTR IS 'C'" and "REL:ATTR IS #N", where C is an identifier and N is a number, indicate that values of attribute ATTR in the relation REL are restricted to the character constant 'C' and the number N, respectively. So they correspond to the formula REL(...,'C'/ATTR,...) and REL(...,#N/ATTR,...) in the multi-layer logic, respectively.



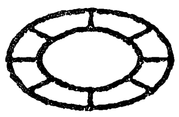
denotes some values of the (multi)set obtained by projecting the relation REL on the attribute ATTR. Thus, it is the direct counterpart of ($\exists$ x/ATTR) [REL(...,x,...)] of the multi-layer logic. On the other hand, 

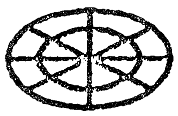
denotes all the values of the (multi)set obtained from the relation REL. Thus, it corresponds to the formula ($\forall$ x/ATTR) [REL(...,x,...)]. Inner ellipse of 

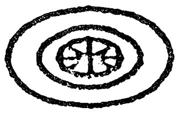
represents a

subset of the (multi)set obtained by projecting the relation REL on the attribute ATTR. In other words, it denotes an example of a subset of the (multi)set obtained by projecting the relation REL on the attribute ATTR. Thus the outer ellipse indicates the powerset of the (multi)set. Hence, it corresponds to the formula in the multi-layer logic $(\exists\ x2/{*}ATTR)(\exists\ x1/x2)\ [\ REL(\ldots,x1,\ldots)\ ]$. 

in the GOING expression is used for representing all the values of some subsets of the (multi)set obtained by projecting the relation REL on the attribute ATTR. Thus it corresponds to a formula $(\exists\ x2/{*}ATTR)(\forall\ x1/x2)\ [\ REL(\ldots,x1,\ldots)\ ]$. In this manner,



and


correspond to $(\forall\ x2/{*}ATTR)(\exists\ x1/x2)\ [\ REL(\ldots,x1,\ldots)\ ]$ and $(\forall\ x2/{*}ATTR)(\forall\ x1/x2)[\ REL(\ldots,x1,\ldots)\ ]$, respectively. These expressions are naturally extended to those representing a powerset of the (multi)set obtained by projecting the relation REL on the attribute ATTR. For example,


corresponds to the formula $(\exists\ x3/{**}ATTR)(\exists\ x2/x3)(\forall\ x1/x2)\ [\ REL(\ldots,x1,\ldots)\ ]$. Table 1 summarizes the correspondence of the GOING expressions.

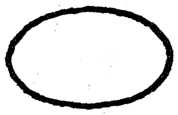
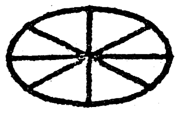
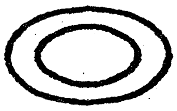
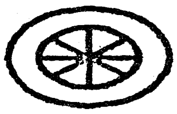
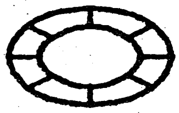
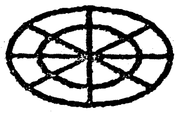
GOING	The multi-layer logic
REL:ATTR IS 'C'	REL(...,'C'/ATTR,...)
REL:ATTR IS #N	REL(...,#N/ATTR,...)
 REL:ATTR	(EX/ATTR) [REL(...,X,...)]
 REL:ATTR	(AX/ATTR) [REL(...,X,...)]
 REL:ATTR	(EX2/*ATTR)(EX1/X2) [REL(...,X1,...)]
 REL:ATTR	(EX2/*ATTR)(AX1/X2) [REL(...,X1,...)]
 REL:ATTR	(AX2/*ATTR)(EX1/X2) [REL(...,X1,...)]
 REL:ATTR	(AX2/*ATTR)(AX1/X2) [REL(...,X1,...)]

Table. 1 Correspondence of GOING to the multi-layer logic.

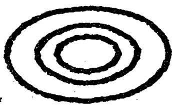
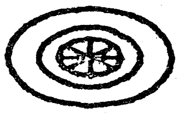
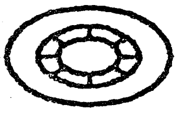
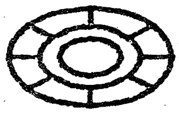
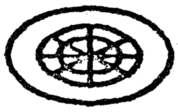
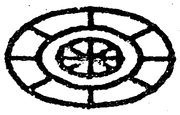
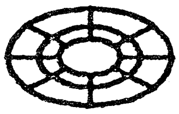
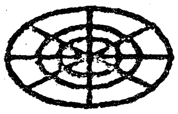
GOING	The multi-layer logic
 REL:ATTR	(EX3/**ATTR)(EX2/X3)(EX1/X2) [REL(...,X1,...)]
 REL:ATTR	(EX3/**ATTR)(EX2/X3)(AX1/X2) [REL(...,X1,...)]
 REL:ATTR	(EX3/**ATTR)(AX2/X3)(EX1/X2) [REL(...,X1,...)]
 REL:ATTR	(AX3/**ATTR)(EX2/X3)(EX1/X2) [REL(...,X1,...)]
 REL:ATTR	(EX3/**ATTR)(AX2/X3)(AX1/X2) [REL(...,X1,...)]
 REL:ATTR	(AX3/**ATTR)(EX2/X3)(AX1/X2) [REL(...,X1,...)]
 REL:ATTR	(AX3/**ATTR)(AX2/X3)(EX1/X2) [REL(...,X1,...)]
 REL:ATTR	(AX3/**ATTR)(AX2/X3)(AX1/X2) [REL(...,X1,...)]

Table. 1 (continued)

4. Implementation of GOING

4.1. Commands for GOING expression

In this section we describe the commands provided by the current GOING implementation. They consists of the following three classes :

(1) definition of variables and their domains,

(2) definition of quantification and logical order of entities,

(3) definition of expressions involving predicate constants and or functions.

For each concept , GOING provides one or more commands. In the following, <c> denotes to locate the cursor where the user desire to define something , and <c/r> indicates to press the carriage return key.

Command 1 : <c> S m <c/r> .

This command indicates to define a set and display a ellipse of size m at the position where the cursor is located. m is an integer not greater than 8. The size of the ellipse to be displayed is determined by the function $x = 5\sqrt{2.6^{m-1}}\cos(\theta)$, $y = 3\sqrt{2.6^{m-1}}\sin(\theta)$ [mm].

Command 2 : <c> P m <c/r> .

This command indicates to define a set and display two ellipses of size m and m-1 at the position where the cursor is located.

Command 3 : <c> T m <c/r> .

This command indicates to define a set and display three ellipses of size m, m-1 and m-2 at the position where the cursor is located.

Command 4 : <c> N s <c/r> .

The command is available for defining a domain name for a set, a power set or a set of power set. s is a string of characters having the syntax <relation name>:<attribute name>.

Command 5 : <c> A <c> E <c/r> .

This command provides for defining universal-existential quantification. That is, for all elements of the domain indicated by the first cursor positioning, there exist at least one element of the domain specified by the second cursor positioning. As the result of excution of this command , a directed arc is displayed, which connects a element of the domain indicated by the first cursor positioning and that of the domain specified by the second.

Three other commands are defined similarly.

Command 6 : <c> E <c> E <c/r> .

This command serves for defining existential-existential quantification.

Command 7 : <c> A <c> A <c/r> .

This command provides for defining universal-universal quantification.

Command 8 : <c> E <c> A <c/r> .

This command serves for defining existential-universal quantification.

Command 9 : <c> D <c> <c/r> .

This command is used to define a disjunctive relation of two directed arcs, one is indicated by the first cursor positioning and the other is specified by the second cursor positioning.

Command 10 : <c> X s <c/r> .

This command indicates to define a expression involving predicate constants and/or functions. The expression to be defined has to be obey the syntax described in the appendix.

As an example, the query 4 illustrated in Section 2 is expressed by means of the following command sequence :

```
<c>  S 4  <c/r>                          ----   (1)
<c>  N LU:USAG  <c/r>                    ----   (2)
<c>  T 6  <c/r>                          ----   (3)
<c>  N LU:LAND  <c/r>                    ----   (4)
<c>  N LDA:LAND  <c/r>                   ----   (5)
<c>  N LYP:LAND  <c/r>                   ----   (6)
<c>  A  <c>  E  <c/r>                    ----   (7)
```

(First, the cursor is pointed inside of the ellipse defined by (1) , next it is on the side of the innermost ellipse specified by (3).)

```
<c>  X LYP:YEAR IS #1981  <c/r>          ----   (8)
<c>  A  <c>  E  <c/r>                    ----   (9)
```

(First, the cursor is pointed inside of the innermost ellipse specified by (3), next it is pointed the first argument of the expression specified by (8), i.e.":".)

```
<c>  X LDA:AREA IS @GE #100  <c/r>      ----   (10)
<c>  A  <c>  E  <c/r>                   ----   (11)
```

(Similar to the command 9.)

```
<c> X ^AVG(UCOUNT(*))  <c/r>            ----   (12)
<c>  A  <c>  E  <c/r>                   ----   (13)
```

(Similar to the command 9.)

However, as mentioned above, the order of commands is immaterial.

4.2. Algorithm to translate a GOING expression into a formula in the multi-layer logic

4.2.1. An overview

Algorithm to translate a GOING expression into a formula in the multi-layer logic is divided into two parts: generation of a prefix and that of a body.

Generation of a prefix includes

(1) to get variables required from a given GOING expression,

(2) to determine the quantifiers and logical order of variables in a prefix.

This algorithm will be discussed in detail in the following sections. On the other hand, algorithm generating a body of a formula is implemented by simple symbol manipulation. The key of this algorithm involves to reference the schema of the relational database concerned, to list up the predicates required, and to assign constants and functions obtained from a GOING expression to the arguments of the predicates. Since the implementation of algorithm generating a body contains no novel techniques , it is not discussed here.

4.2.2. To generate variables from a GOING expression

Variables are generated according to the following rules.

(VG-1) For a domain specified by

(1) command 1, a variable defined on the domain is generated ;

(2) command 2, two variables are generated, i.e., a variable defined on the set of subsets of the domain, say X, and a variable defined on the X ;

(3) command 3, three variables are generated,i.e., a variable defined on the set of powersets of the domain, say XX, and a variable defined on the XX , say X, and a variable on the X.

(VG-2) For an expression containing a predicate constant, and the arguments take the form of <relation name>:<attribute name> , a variable is generated.

(VG-3) For a function whose value is to be answered, a variable is generated.

4.2.3. To generate priority informations

Another information to be produced is that of priority. This information is obtained from the following rules :

(PI-1) For the variable generated by the rule (VG-1) the priority information is N*100 , where N is an integer corresponding to the level of the variable i.e. N=1 for command 1, N=2 for command 2, N=3 for command 3.

(PI-2) For the variable generated by the rule (VG-2) this information is 100 ;

(PI-3) For the variable generated by the rule (VG-3) the priority is 100 + 'the priority information of the output variable', which is set 99 at first and is decremented by one for each time an output variable is generated.

4.2.4. To generate a matrix for the logical order of variables

The variable matrix discussed in this section is a matrix that is used to determine the logical order of variables in the prefix of a formula. The logical order of variables are expressed in the GOING expression by means of directed arcs and the structure of domains. The former is the logical order explicitly expressed by directed arcs. While the latter implicitly indicates the logical order of a variable whose value is set and a variable defined over it. The value of an element of the variable matrix e(i,j) is defined as follows :

e(i,j) = N , if the j-th variable precedes to the i-th variable, where N is an integer corresponding to the level of the j-th variable ;

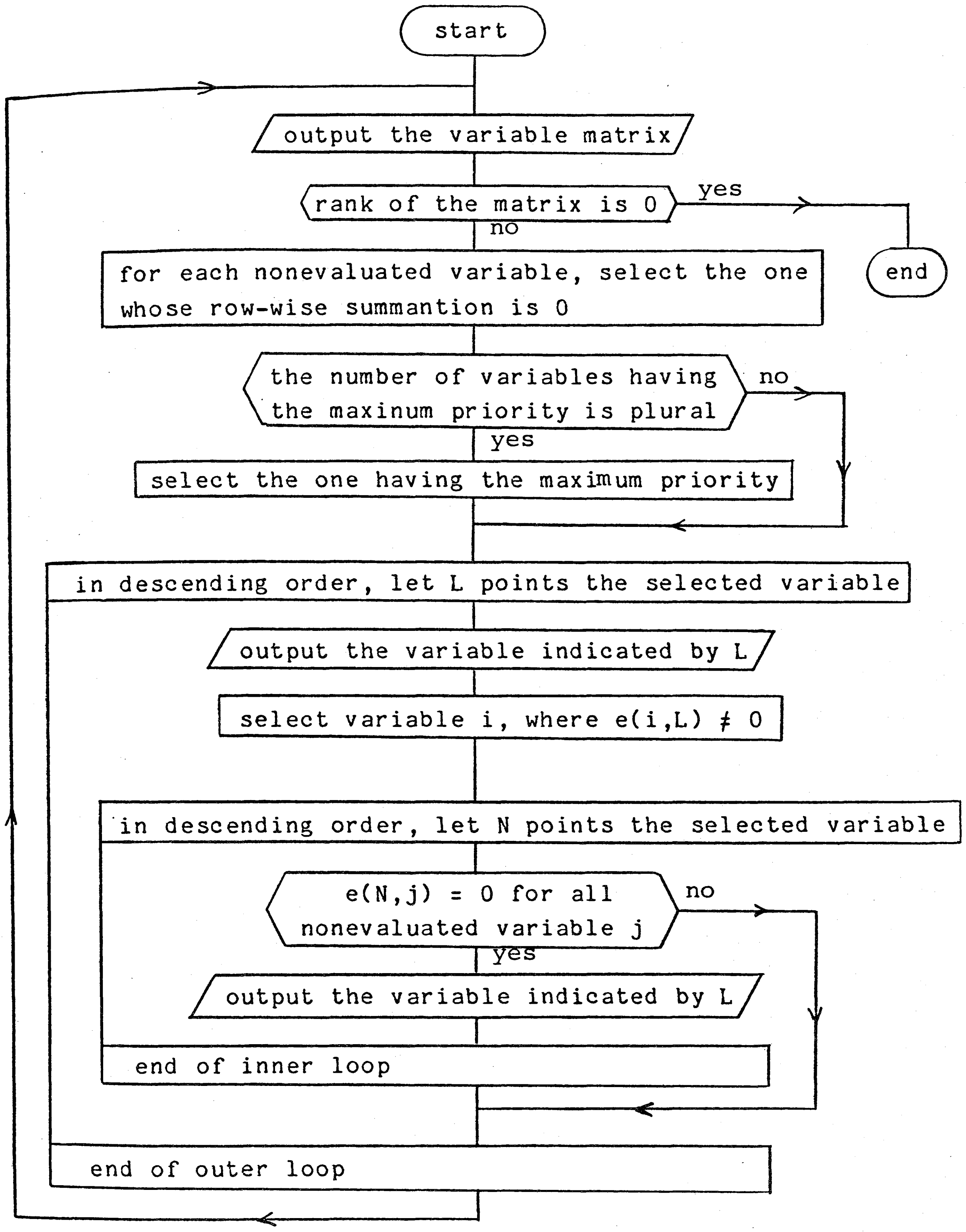
e(i,j) = 0 , otherwise.

As an example, the variable matrix shown in Figure 1 (pre-

```
1 1020  250  400  5  0  1  0  0  0  0  0  0     0     0    0    0
2   11    0    0  0  0  0  0  0  0  0  0  0  5052     0    0    0
3 1040  600  400  6  0  2  3  4  0  0  0  0     0     0    0    0
4   30    0    0  0  0  0  0  0  0  0  0  0  7102     0    0    0
5   20    0    0  0  0  0  0  0  0  0  0  0     0     0    0    0
6   11    0    0  0  0  0  0  0  0  0  0  0  5103  6103    0    0
1  210  310  LU:USAG
2  560  280  LU:LAND
3  560  260  LDA:LAND
4  560  240  LYP:LAND
5  350  540  LYP:YEAR IS #1981
6  550  650  LDA:AREA IS @GE #100
7  670  600  ^AUG(UCOUNT(*))
```

```
        Z1 Y3 Y2 Y1 H  I
100 Z1  0  0  0  0  0  0
300 Y3  0  0  0  0  0  0
200 Y2  1  2  0  0  0  0
100 Y1  0  0  1  0  0  0
199 H   0  3  0  0  0  0
100 I   0  0  0  1  0  0
```

Figure 1. The variable matrix (preceded by the data structure constructed in the system) obtained from the GOING expression for query 4.



start
output the variable matrix
rank of the matrix is 0
yes
no
end
for each nonevaluated variable, select the one whose row-wise summantion is 0
the number of variables having the maxinum priority is plural
no
yes
select the one having the maximum priority
in descending order, let L points the selected variable
output the variable indicated by L
select variable i, where e(i,L) ≠ 0
in descending order, let N points the selected variable
e(N,j) = 0 for all nonevaluated variable j
no
yes
output the variable indicated by L
end of inner loop
end of outer loop

Figure 2. General flowchart of prefix generation.

ceded by the data structure constructed in the system) is obtained from the GOING expression for query 4 in Section 2. The leftmost column represents priority informations corresponding to the variables in the second column.

4.2.5. To generate a prefix

The general flowchart for generating a prefix is given in Figure 2. The result of the application of this algorithm to the vari able matrix given in Figure 1 is shown in Figure 3.

```
(E Y3/**LAND)
(E Y3/**LAND)(E H ^/REAL)
(E Y3/**LAND)(E H ^/REAL)(A Z1/USAG)
(E Y3/**LAND)(E H ^/REAL)(A Z1/USAG)(E Y2/Y3)


          Y1  I
 197 Y1   0   0
 100 I    1   0


(E Y3/**LAND)(E H ^/REAL)(A Z1/USAG)(E Y2/Y3)(A Y1/Y2)
(E Y3/**LAND)(E H ^/REAL)(A Z1/USAG)(E Y2/Y3)(A Y1/Y2)
(E I /AREA)


**  Well-formed formula reduced from a GOING expression  **

(E Y3/**LAND)(E H ^/REAL)(A Z1/USAG)(E Y2/Y3)(A Y1/Y2)
(E I /AREA)
[
   LYP( Y1, #1981/YEAR, @   )
 & LU( Y1, Z1 )
 & LOA( Y1, @ , I   )
 & GE( I, #100/AREA )
 & LET( H , AVG(UCOUNT(Y3)) )
]
```

Figure 3. The result of the application of the translation algorithm to query 4.

ACKNOWLEDGEMENTS

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

I am grateful to S.Kunifuji, IIAS of Fujitsu Limited, for many helpful comments and providing 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. "On the application of the multi-layer logic to a relational database query language," (in Japanese) WGDB Meeting of IPSJ 25-1 (1981).

3] Udagawa, Y. and Ohsuga,S. "The multi-layer logic as a relational database query language and its reduction algorithm to relational procedures," (in Japanese), Joho-Shori (submitted).

4] 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.

5] Udagawa,Y. and Ohsuga,S. "Construction of SBDS-F3 : a relational database with inference mechanism," RIMS, Univ of Kyoto, Kokyu-Roku, 1982.

6] McDonald,N. "CUPID -- A graphics oriented facility for support of non-programmer interactions with a data base," ERL, Univ. Calif. Berkeley, Mem #ERL-M563, (Novem. 1975).

7] Zloof,M.M. "Query-by-Example : a data base language," IBM Syst.J.4, pp324-343(1977).

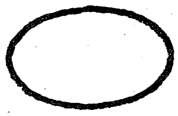
APPENDIX

Syntax of the GOING expression:

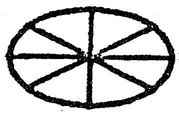
The goals of the GOING syntax are simplicity, intuitiveness, ease of use. In the sequel the syntax of the query language GOING is given. The syntax is described by the Backus notation. In the following, the special symbols x => y , s $\textcircled{or}$ t are used for expressing connection from an expression x to y by a directed arc and connection from a directed arc s to t by a undirected arc, respectively.

```
<GOING expression> ::= <domain specification> ¦
                       <literal expression> ¦
                  <GOING expression><arc><domain specification> ¦
                  <GOING expression><arc><literal expression>
```

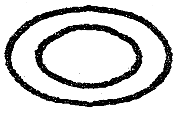
<domain specification> ::=



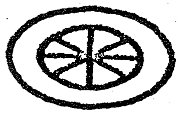
<domain name> ¦



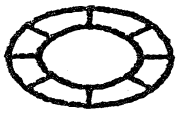
<domain name> ¦



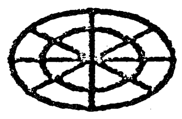
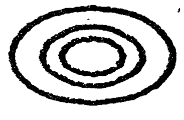
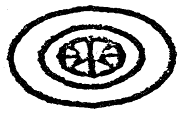
<domain name> ¦



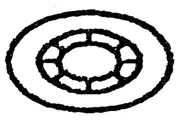
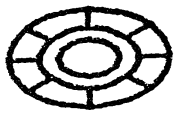
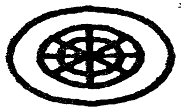
<domain name> ¦



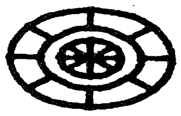
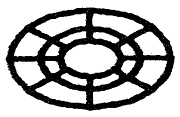
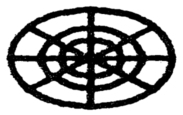
<domain name> ¦

 ¦  ¦  ¦

<domain name>   <domain name>   <domain name>

 ¦  ¦  ¦

<domain name>   <domain name>   <domain name>

 ¦  ¦  ¦

<domain name>   <domain name>   <domain name>

<arc> ::= => ¦ => $\textcircled{or}$ =>

<literal expression> ::= <Boolean expression> ¦
^<functional expression> ¦

<Boolean expression> ::= <argument> IS <argument> ¦
<argument> IS
@<predicate constant><argument>

<functional expression> ::= <function name>
(<functional argument list>)

<functional argument list> ::= <functional argument>,
<functional argument list>

<functional argument> ::= * ¦ <constant>

<constant> ::= #<number> ¦ '<identifier>'

<argument> ::= * ¦ <domain name> ¦ <functional expression>

<domain name> ::= <relation name> : <attribute name>
¦ ^<relation name> : <attribute name>

```
<predicate constant> ::=  EQ ¦ NE ¦ GT ¦ LT ¦ GE ¦ LE ¦ SSET
                        ¦ DISJ
<function name> ::=  add ¦ sub ¦ mult ¦ div ¦ mod
                   ¦ ucount ¦ count ¦ summ ¦ avg .
<function name> ::= <identifier>
<relation name> ::= <identifier>
<attribute name> ::= <identifier>
<identifier> ::= <letter> ¦ <identifier><letter> ¦
                <identifier><digit>
<number> ::=  <unsigned number> ¦ +<unsigned number> ¦
             -<unsigned number>
<unsigned number> ::= <decimal number> ¦
                      <decimal number> E<integer>
<decimal number> ::= <unsigned integer> ¦ <decimal fraction> ¦
                     <unsigned integer><decimal fraction>
<unsigned integer> ::= <digit> ¦ <unsigned integer><digit>
<decimal fraction> ::= .<unsigned integer>
<integer> ::=  <unsigned integer> ¦ +<unsigned integer> ¦
              -<unsigned integer>
<letter> ::= A¦B¦C¦D¦E¦F¦G¦H¦I¦J¦K¦L¦M¦N¦O¦P¦Q¦R¦S¦T¦U¦V¦W¦X¦Y¦Z
<digit>  ::= 1¦2¦3¦4¦5¦6¦7¦8¦9¦0
```