Novel Technique to Interact with Relational Databases by Using a Graphics Display

YOSIHISA UDAGAWA* and SETSUO OHSUGA*

The widespread dissemination of information systems for non-programmers requires a database interface language that has high descriptive power and permits users to interact with the systems in other than computer oriented terms. This paper provides the design and implementation of a non-procedural user interface language for relational databases which is named GOING. Taking advantage of two-dimensional representation by using graphic displays, GOING provides a concise, easy-to-understand representation of queries. Queries are expressed in terms of simple figures, i.e. ellipse for domains, arcs for logical orders of entities and connection of conditions, as well as expressions composed of comparison predicates and functions. With these techniques we can avoid difficult natural language processing by the system and spelling mistakes by users while retaining high descriptive power.

In this paper, GOING expressions are compared with two other graphics oriented languages, i.e. CUPID and Query-by-Example. It is shown that GOING has high descriptive power and is systematic in expressing queries. GOING also has strong theoretical basis. Any GOING expression has a counterpart in an extended many-sorted logic and the correspondence between GOING and this logic is discussed. Implementation of a translation algorithm from GOING expression into a formula in this logic is also described.

1. Introduction

In order to use database systems in more widespred applications, it is essential to develop data languages that have high descriptive power and permit users to interact with the systems not in computer oriented terms [1, 6]. For these requirements, we designed and implemented a non-procedural user interface language for relational databases which is named GOING (a **G**raphics **O**riented **IN**teractive data lan**G**uage).

Two approaches are commonly known to achieve a user-friendly interface language. One is to describe requests by means of (restricted) natural language. However, this approach meets only limited queries because natural language processing includes difficult syntactical and semantical analysis [2]. The other is to express a query by means of a graph. By taking advantage of two-dimensional representation by using a graphics display, queries are expressed within a simple and easy-to-understand conceptual framework.

GOING 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 languages with a similar basic orientation include CUPID [4] and Query-by-Example [10].

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, arcs for logical orders of entities and connection of conditions), and expressions composed of comparison predicates and functions.

(2) GOING uses very simple English-like text. Thus, we can avoid difficult natural language processing by the system and spelling mistakes by users.

(3) 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.

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

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

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

(7) GOING has strong theoretical basis. A query expression in GOING is translated into an intermediate language, (the multi-layer logic for relational databases [5, 7, 8], and then reduced to a sequence of high level procedural operations of relational algebra.

(8) GOING provides high expressive power. GOING is able to describe any formula in the multi-layer logic for relational databases, which is proved to have more expressive power than conventional query languages based on the predicate logic and graphics-oriented query languages [5, 6, 7].

In Section 2, we introduce GOING and compare it with

*Institute of Interdisciplinary Research Faculty of Engineering, Tokyo University 4-6-1 Komaba, Meguro-ku, Tokyo 153, Japan.

two other graphics oriented languages, i.e. CUPID and Query-by-Example. In Section 3, we discuss informally the multi-layer logic for relational databases and provide correspondence between GOING expressions and basic concepts in this logic. Section 4 deals with an algorithm to translate a GOING expression into a formula in the multi-layer logic. Section 5 concludes this paper. A BNF syntax for GOING is given in the Appendix.

2. GOING Expressions for Relational Database Queries

2.1. GOING Expressions for Basic Database Queries

In this section, basic queries of the relational database are illustrated in the GOING expressions. A domain or literal expression preceded by "$\wedge$" symbol means that it's the value will be listed.

(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, the GOING expression in Fig. 1 illustrates how to retrieve the values of ATTR1 in the relation REL from the relation whose associating values of ATTR2 satisfy the given condition.

(B) To Express Queries Containing Aggregation Functions

Many 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 GOING expression in Fig. 2.

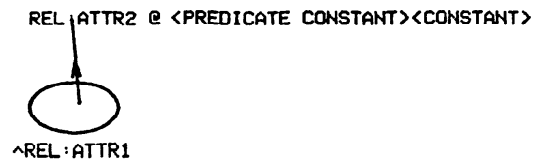


Fig. 1 GOING expression for queries containing Boolean conditions.

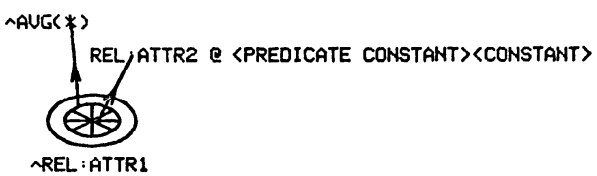


Fig. 2 GOING expression for queries containing aggregation functions.

(C) To Express Queries Containing "Group by" Operations

In some cases, queries involve aggregation functions whose arguments are determined for each element of some other set (i.e. for all elements of some set, there exist subsets of a set, which are arguments of a given aggregation function). These kinds 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 GOING expression in Fig. 3.

2.2. Describing Queries in Terms of GOING Expressions and Comparing GOING with CUPID and Query-by-Example

In this section, we illustrate GOING expressions for rather complex queries and compare them with other graphics oriented languages, i.e. CUPID and 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 the price and year for each piece of land. The relation LU gives the usage of each piece of land. The relation LDA gives, for each land, its area and 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 and their areas with usage 'A', which are less than 35 kilometer apart from the center of a city and whose prices are less than 600,000 YEN/m^2 in the year 1981.

Figure 4 contains three expressions for query 1, i.e. expressions in GOING, CUPID and Query-by-Example. In the GOING expression, an arc which connects 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 and LDA whose usages are 'A'. The literal expressions LYP: PRIC IS $\sim$LT #60 and LDA: DIST IS $\sim$LT #35 denote that the values of the price in the relation

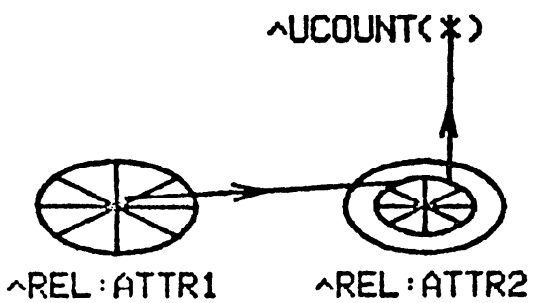


Fig. 3 GOING expression for queries containing "group by" operations.

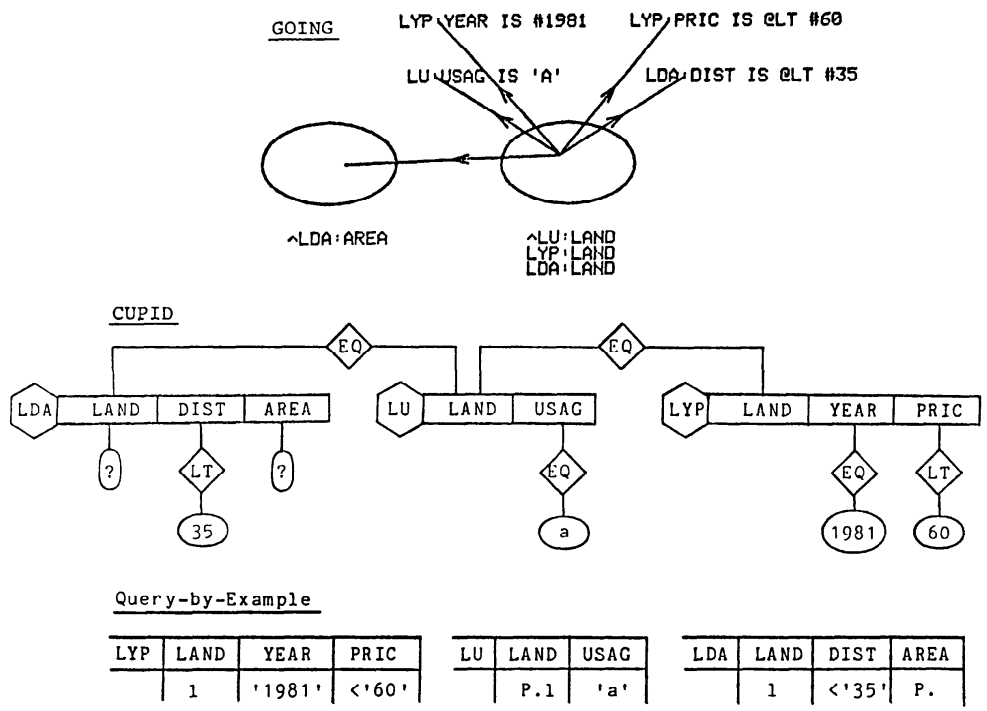


Fig. 4 Query 1 in GOING, CUPID and Query-by-Example.

LYP and the values of the distance in the relation LDA are less than 60 and 35.

A query against more than one relation with a universal quantification

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

Note that only those domains referred to need be represented in the GOING expression in Fig. 5. In this example the column PRIC in the relation LYP and column AREA in the relation LDA are not used in the GOING expression.

A query with built-in arithmetic and aggregation functions

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

In the GOING expression in Fig. 6, the literal expression SUB(*, AVG(*)) demonstrates that a built-in function may be nested to any level 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, while implicitly expressed in CUPID. According to [3, 10], this query is not expressible in Query-by-Example.

A query with a nested aggregation function and Boolean conditions

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 meters.

The GOING expression in Fig. 7 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 meters. 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. In CUPID, on the other hand, the concepts of subsets of a set, etc. are expressed in the same syntax. According to [3, 10], this query is not expressible in Query-by-Example.

3. Multi-Layer Logic for Relational Databases and GOING Expressions

3.1. Multi-Layer Logic as an Extension of the Many-Sorted Logic

It is recognized that the first order many-sorted logic is not broad enough in expressing queries for the relational database [3, 5, 6]. Many of the practical queries that contain aggregation functions and/or group-by

Figure 5. Query 2 in GOING, CUPID and Query-by-Example.

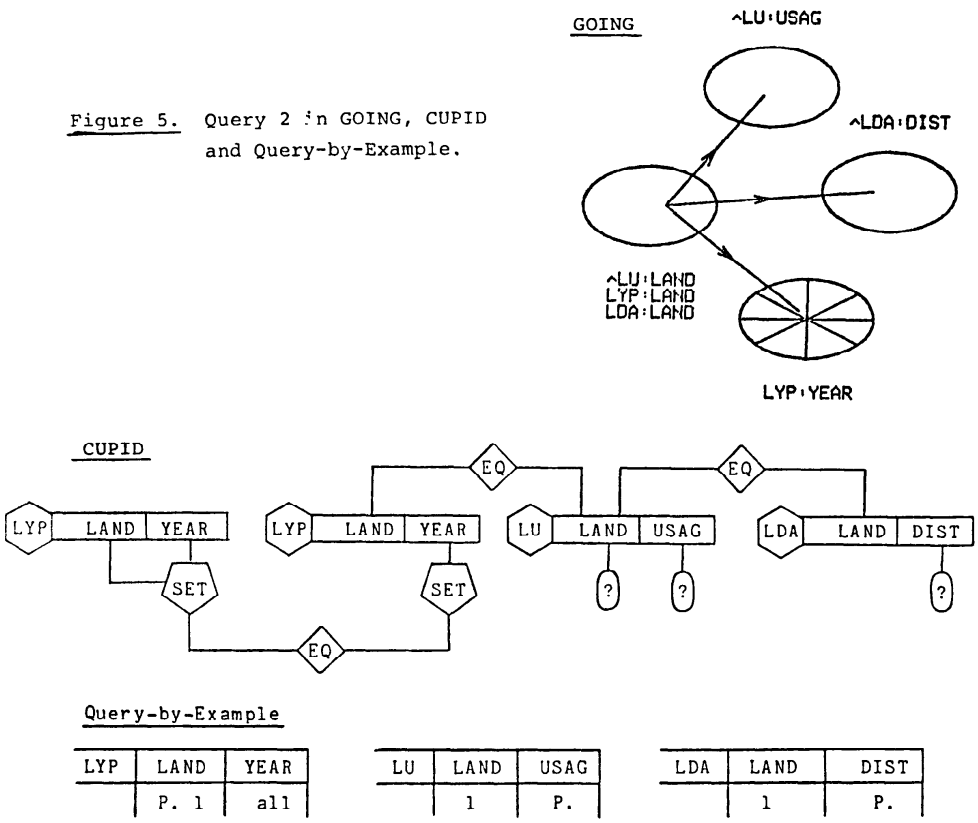


Fig. 5 Query 2 in GOING, CUPID and Query-by-Example.

operations cannot be expressed by formulas in the many-sorted logic. The approach that we have taken is to extend the many-sorted logic. In the many-sorted logic, the type of a variable is restricted to the predefined types, e.g. integer, real, and so on. In the multi-layer logic discussed in this paper, "set", a method of structuring types is introduced. A power set of an arbitrary set S, excluding the empty set, is denoted by *S. For example, let S′ be a finite set of natural numbers, and G(x, y) be a predicate which is defined as true when x is greater than y, and as false otherwise. Let SO(X) denote a predicate containing a one-place set operation. Then the formula $(\exists X/{*S'})(\forall x/X)G(x, 50)$ & SO(X) denotes a query demanding to calculate SO(X) to some X which is a subset of S′ (or element of *S′) and each element of S′ satisfies the predicate G(x, 50). This idea is naturally extended to power sets, power-power sets and so on. If $x_0, x_1, \cdots, x_n$ $(n \geq 0)$ are variables defined over S, $*S, \cdots, *_n \cdots *_1 S$, respectively, then $(Q_n x_n/*_n \cdots *_1 S)(Q_{n-1} x_{n-1}/x_n) \cdots (Q_0 x_0/x_1)$ is a prefix of the multi-layer logic, where $Q_i (0 \leq i \leq n)$ is either $\forall$ or $\exists$. When $n=0$, a prefix is $(Q_0 x_0/S)$, which is the prefix of the many-sorted logic. The multi-layer logic is a kind of higher order logic, but only one-place predicates are allowed to be variables. The relationship between this logic and the conventional higher order logic can be best illustrated by Fig. 8. For details, [7, 8] can be referred.

3.2. Correspondence Between GOING Expression and Basic Concepts in the Multi-Layer Logic for Relational Databases

In this section, correspondence between GOING expression and basic concepts in the multi-layer logic is discussed. A formula in this logic is constructed in terms of;

(M-1) relationships between a variable and its domain,
(M-2) quantification of variables,
(M-3) Predicate constants possibly involving 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 expres-

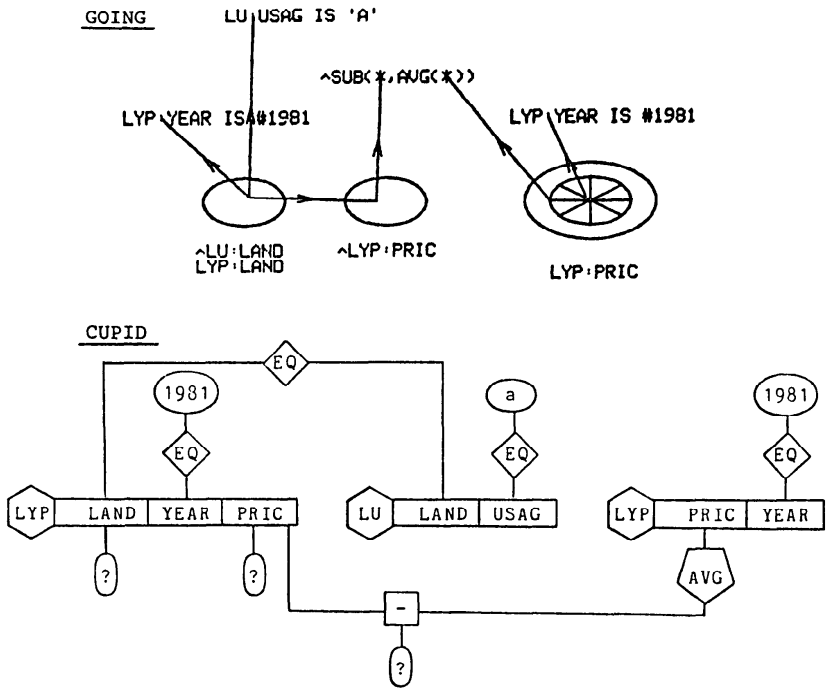


Fig. 6 Query 3 in GOING and CUPID.

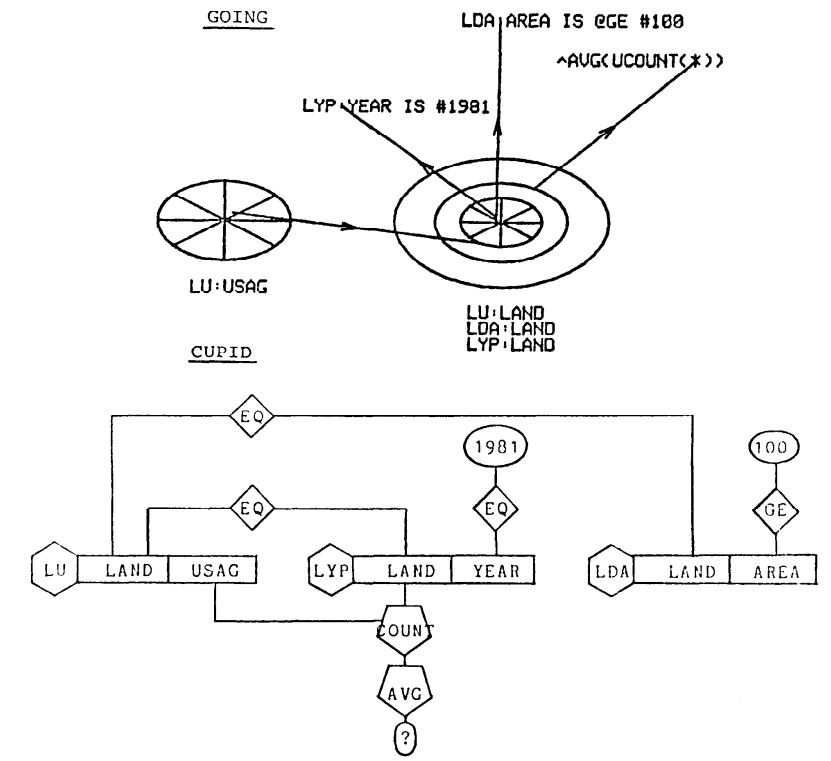


Fig. 7 Query 4 in GOING and CUPID.

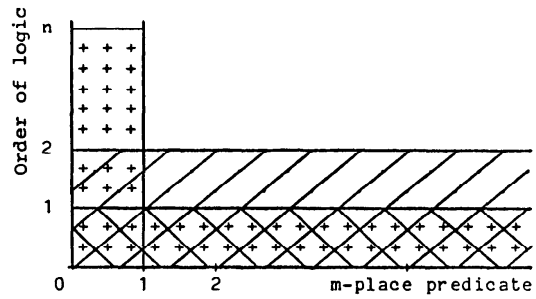


Fig. 8 The relationship between the multi-layer logic and the conventional logic. + + +, \\\, /// indicate the multilayer logic, 1-st order logic, 2-nd order logic, respectively.

sions,

(G-4) directed arcs,

(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→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. They correspond to the formula REL(···, 'C'/ATTR,···) and REL(···, #N/ATTR,···) in the multi-layer logic, respectively.  denotes some values of the set obtained by projecting the relation REL on the attribute ATTR. Thus, it is the direct counterpart of (∃ x/ATTR) [REL(···, x,···)] of the multi-layer logic. On the other hand,  denotes all the values of the set obtained from the relation REL. Thus, it corresponds to the formula (∀ x/ATTR) [REL(···, x,···)]. Inner ellipse of  represents a subset of the set obtained by projecting the relation REL on the attribute ATTR. In other words, it denotes an example of a subset of the set obtained by projecting the relation REL on the attribute ATTR. Thus the outer ellipse indicates the powerset of the set. Hence, it corresponds to the formula in the multi-layer logic (∃ x2/*ATTR)(∃ x1/x2) [REL(···, x1,···)].  in the GOING expression is used for representing all the values of some subsets of the set obtained by projecting the relation REL on the attribute ATTR. Thus it corresponds to a formula (∃ x2/*ATTR)(∀ x1/x2) [REL(···, x1,···)]. In this manner,  and  correspond to (∀ x2/*ATTR)(∃ x1/x2) [REL(···, x1,···)] and (∀ x2/*ATTR)(∀ x1/x2) [REL(···, x1,···)], respectively. These expressions are naturally extended to those representing a powerset of the set obtained by projecting the relation REL on the attribute ATTR. For example, 

Table 1 Correspondence between GOING and multi-layer logic.

GOING	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,...)]
 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)(AX2/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,...)]

corresponds to the formula (∃ x3/**ATTR)(∃ x2/x3)(∀ x1/x2) [REL(···, x1, ···)]. Table 1 summarizes the correspondence between GOING expressions and formulas in the multi-layer logic for relational databases.

4. Algorithm to Translate a GOING Expression into a Formula in the Multi-Layer Logic for Relational Databases

4.1. Overview

Algorithm to translate a GOING expression into a formula in the multi-layer logic is generally divided into five parts:

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

(2) to generate priority information of variables,

(3) to generate a matrix to determine the logical order of variables,

(4) to generate a prefix,

(5) to construct a body of a formula.

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

4.2. To Get Variables From a GOING Expression

Variables are generated according to the following rules.

(VG-1) For a domain represented by simple ellipse, a variable defined on the domain is generated, say X1;

(VG-2) For a domain represented by doubl-ellipse, two variables are generated, i.e. a variable defined on the set of subsets of the domain, say X2, and a variable defined on the X2, say X1;

(VG-3) For a domain represented by triple-ellipse, three variables are generated, i.e. a variable defined on the set of powersets of the domain, say X3, a variable defined on the X3, say X2, and a variable on the X2, X2, say X1.

(VG-4) For an expression containing a predicate constant, and the arguments take the form of ⟨relation name⟩: ⟨attribute name⟩, a variable is generated.

(VG-5) For a function whose value is to be answered, a variable is generated.

4.3. To Generate Priority Information of Variables

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

(PI-1) For the variables generated by the rules (VG-1), (VG-2) and (VG-3), the priority information are N*100, where N = 1 for (VG-1), N = 2 for (VG-2) and N = 3 for (VG-3).

(PI-2) For the variable generated by the rule (VG-4) this information is 100;

(PI-3) For the variable generated by the rule (VG-5) the priority is 100+ 'the priority information of the output varialbe', which is set at 99 at first and is decremented by one for each time an output variable is generated.

4.4. To generate a Matrix To determine 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 variable matrix represents both these logical orders of variables. 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 the i-th variable, where N is an integer obtained by 'priority information of the j-th variable' ÷ 100; e(i, j) = 0, otherwise.

As an example, the variable matrix shown in Fig. 9 (preceded 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 information corresponding to the variables in the second column.

4.5. To Generate a Prefix

Algorithm for generating a prefix is given below.

Step 1. If the rank of the matrix is 0 then stop, else go to Step 2.

Step 2. Get a set **V** of non-evaluated variables whose row-wise summation is 0, i.e. a set of a variables preceded by no variables. If **V** is empty then a given GOING expression is illegal, else go to Step 3.

Step 3. If the set **V** is empty then go to Step 1, else select a variable L having maximum priority from **V** and go to Step 4.

Step 4. Print the variable L. The variable L is evaluated. Get a set **W** of non-evaluated variables i which satisfy e(i, L) ≠ 0. Go to Step 5.

Step 5. If the set **W** is empty then go to Step 3, else select a variable M having maximum priority from **W** and go to Step 6.

Step 6. If e(M, j) = 0 for all non-evaluated variable j then print the variable M. The variable M is evaluated. Go to Step 3.

The result of the application of this algorithm to the variable matrix given in Fig. 9 is shown in Fig. 10. Typical GOING expressions are translated into corresponding formulas in the multi-layer logic in a second.

```
 1 1020  250  400  5  0  1  0  0  0  0  0  0    0    0
 2   11    0    0  0  0  0  0  0  0  0  0  0 5052    0
 3 1040  600  400  6  0  2  3  4  0  0  0  0    0    0
 4   30    0    0  0  0  0  0  0  0  0  0  0 7102    0
 5   20    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
 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  ^AVG(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
```

Fig. 9 Data structure and variable matrix for query 4 in Section 2.

```
(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 )
 & LDA( Y1, @ , I  )
 & GE( I, #100/AREA )
 & LET( H , AVG(UCOUNT(Y3)) )
]
```

Fig. 10 The translated formula in the multi-layer logic for query 4 in Section 2.

5. Conclusion

This paper concerns design and implementation of a graphics oriented data landauage GOING. It is shown that, by taking advantage of two-dimensional representation, queries are expressed within a simple and easy-to-use conceptual framework. GOING expressions are compared with two other graphics oriented languages, i.e. CUPID and Query-by-Example. It is illustrated that GOING has high expressive power and is systematic in expressing queries. A GOING expression has a theoretical counterpart in an extended many-sorted logic, termed the multi-layer logic for relational databases [7, 8]. The correspondence between GOING and this logic is discussed. Implementation of a translation algorithm from a GOING expression to a formula in this logic is also described. It is known that some of the GOING expressions have direct counterparts in relational algebra. For the subject, [7, 9] can be referred. Three future research works are summarized here: explanation facilities for sophisticated user interface, study on database system to manage data with complex data structures, design of total database system with integrity and concurrency control.

Acknowledgements

I wish to acknowledge all the members of Meetings of Information Systems for their encouragement 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 Laboratory of Kyoto University, for providing the SAFE editor system. The SAFE system is very useful and exclusively used for preparing this paper.

References

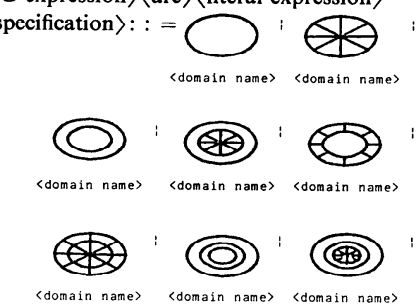
1. BELL, J. R. Future directions in computing, *Computer Vesign* (March, 1981), 95–102.
2. HENDRIX, G. G. etc. Developing a natural language interface to complex data, *ACM Trans. Database Syst.* 3, 2 (June, 1978), 105–147.
3. LACROIX, M. and PIROTTE, A. Example queries in relational languages, *MBLE Technical Note N107* (Sept., 1977).
4. 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).
5. 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).
6. 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) 31–42.
7. UDAGAWA, Y. A Study on Design and Implementation of a a Database System Based on Predicate Logic, *Doctorial Thesis*, Tokyo University (Feb., 1982).
8. UDAGAWA, Y. and OHSUGA, S. Construction of SBDS-F3: a relational database with inference mechanism, *RIMS*, Univ of Kyoto, Kokyu-Roku #461 (1982), 49–78
9. UDAGAWA, Y. and OHSUGA, S. GOING—A data sublanguage using a graphics display, *WGDB Meeting of IPSJ* 29-3 (1982).
10. ZLOOF, M. M. "Query-by-Example: a data base language, *IBM Syst. J.* 4 (1977), 324–343.

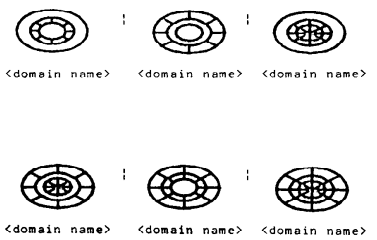
Appendix

In the sequel the syntax for the data language GOING is given. The syntax is described by the Backus notation. In the following, the special symbols x⇒y, s (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.

⟨GOING expression⟩: : =⟨domain specification⟩|
⟨literal expression⟩|
⟨GOING expression⟩⟨arc⟩⟨domain specification⟩|
⟨GOING expression⟩⟨arc⟩⟨literal expression⟩

⟨domain specification⟩: : =





⟨arc⟩: : =⇒|⇒(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

(Received March 29, 1982)