PROGRAPH AS AN ENVIRONMENT FOR PROLOG DATA BASE APPLICATIONS

by T. Pietrzykowski
Acadia University
Wolfville, Nova Scotia
Canada BOP IXO

ABSTRACT

This Paper presents a specialized data base model of a newly developed functional programming language with graphical front PROGRAPH, and discusses the advantages of using it as a host from PROLOG. On the basis of an example there is a description of a method of converting (via PROLOG) an arbitrary query into a wff containing only data base and computational predicates. Afterwards such a wff is formulated in PROGRAPH and computed (again a series of examples and a method provided).

1. Introduction

Logic programming approach proves, beyond any doubt, to be the most universal and flexible mechanism for data base queries ([6], [7] etc.). Its power becomes particularly visible when a query involves a more advanced conceptual structure and requires non trival computations. However, for the full success of logic programing in this area is handicapped by the following shortcomings:

(i) awkwardness of executing queries, which involve universal quantifcation. The usual method of converting them into negated existential qualification of negated formula ([5]) leads often to considerable inefficiency of the search procedure. Moreover, universal

qualification generally accompanies implication inside of the query what in term produces non Horn clauses.

(ii) communication with data base only via unification of the unit clauses brings various dilemnas: the unit clauses with a small number of variables increase flexibility in formulating queries but also increase depth of deduction while unit clauses with a large number of variables may dangerously expend the size of the data base representation and conseqently leads to the lose of efficiency.

(iii) relational data base model (which becomes a consequence of the unit clause approach discussed in (ii)) may causes lose of important information about the overall structure of data base which could be used as valuable heuristical guidelines for an efficient search procedure which is the foundation of the computation of queries.

In the following we shall propose an alternative approach which deals with the data base model as well as with the computation of queries. It will be based on a newly developed programming language PROGRAPH ([3], [4]) which will be used as a host system for logic programing, particularly in the area of interface with a data base.

2. PROGRAPH Data Base.

We will start our presentation from the description of the data base model. It can be viewed as a combination of a particular case of the relational model with some network model influences. However, these similarities may be more misleading then helpful and the best way would be to consider it on its own.

We shall present three different but strictly equivalent definitions of the model. Later we shall refer to any of them: whichever appear to be most convenient.

A. Directed graph model.

The PROGRAPH data base can be defined as a directed graph with labelled arcs. We shall call it graph database or shortly GD. We assume that the graph is connected. The set of nodes N of GD is partitioned into two distinct categories: abstract records and data

records. While abstract records are abstract elements without any particular qualities the data records are trees where leaves are elementary data of basic types like integer, real, boolean, strings of characters, etc.

Among the abstract records there is one specifically distinguished called root and denoted as  . The arcs GD are labelled by a string of characters which are called attributes.

B. Functional Model.

This model is called functional database of FD. It consists of a set elements identical to nodes N of GD and a set of partial multivalued functions with domains and ranges in N. Each of these functions corresponds to a distinct attribute of GD in a one-to-one manner: if F is a function of FD then the domain of F is a subset of N consisting of all nodes where any arc of GD with the attribute F originates while range is the set of all nodes where such an arc ends. The connectivity condition can be easily expressed in terms of FD.

C. Relational Model.

This model called relational database (RD) is a simple variation of the functional model. It is defined as a set of partial binary relations over N. If P is a relation of RD then P(x,y) holds iff y P(x) where P(x) is defined in terms of FD. (The notational ambiguity is hopefully resolved by the reader).

Now we shall present an example of a PROGRAPH database using both GD and RD models. Construction of the FD model for this example is left for the reader

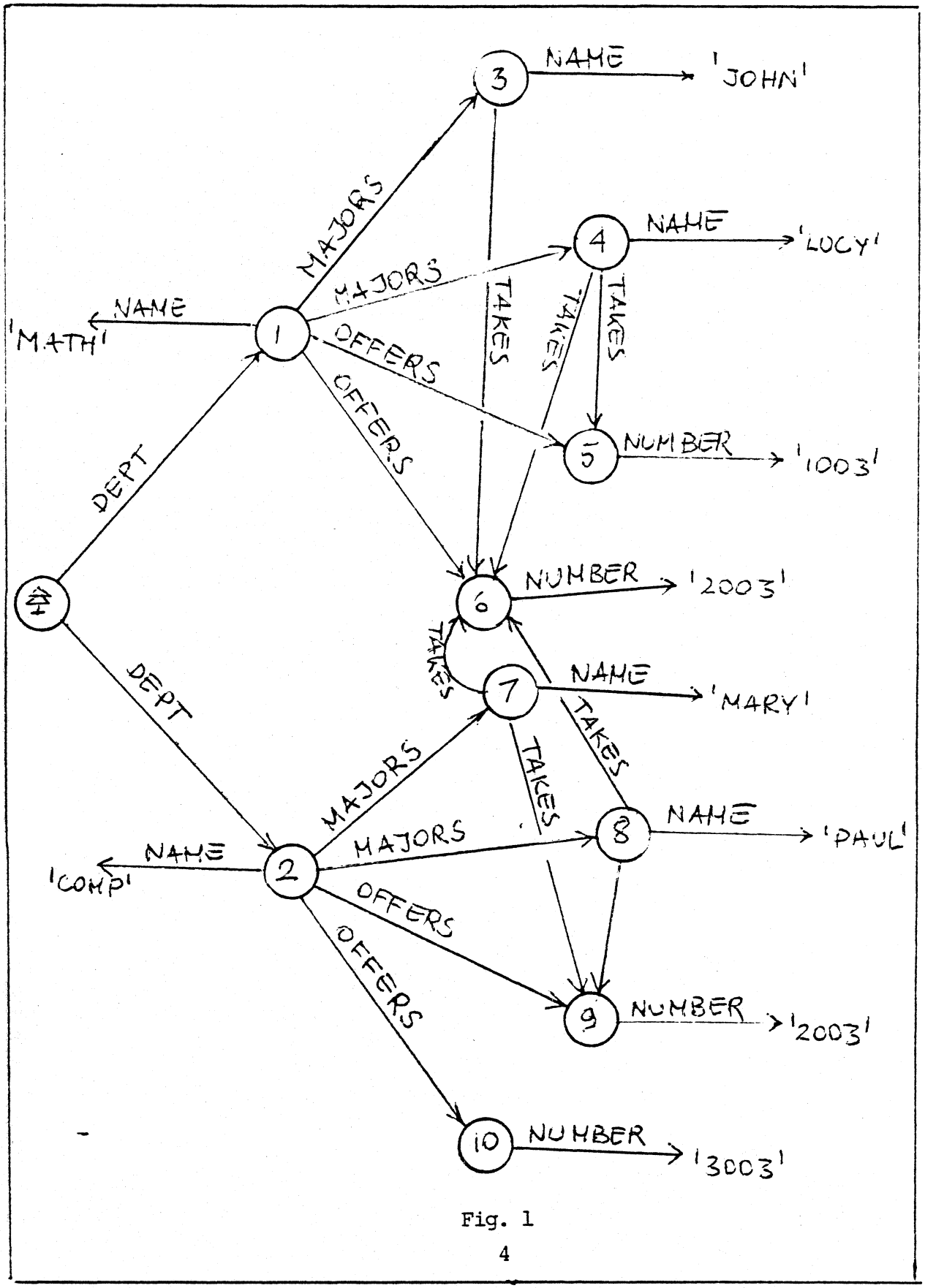

3
NAME
'JOHN'
MAJORS
4
NAME
'LUCY'
MAJORS
NAME
'MATH'
1
TAKES
TAKES
TAKES
OFFERS
OFFERS
5
NUMBER
'1003'
DEPT
OFFERS
NUMBER
'2003'
6
TAKES
7
NAME
'MARY'
DEPT
TAKES
TAKES
MAJORS
MAJORS
8
NAME
'PAUL'
NAME
'COMP'
2
OFFERS
OFFERS
9
NUMBER
'2003'
10
NUMBER
'3003'

Fig. 1

This GD represents a structure composed of departments, students majoring, courses offered, courses taken as well as the names of departments and students and numbers of courses. The integers inside of nodes have no semantic significance and are used only as a way of referring to individual abstract records nodes in further discussion. For example: nodes 1, 2 correspond to departments: 1 to Mathematics while 2 to Computer Science 3, 4, 7, 8 are students with names JOHN, LUCY, MARY and PAUL respectively. The data records have no identifying numbers and we refer to them via the corresponding data.

A useful way of looking at the above GD is to compress it into a map as presented below.

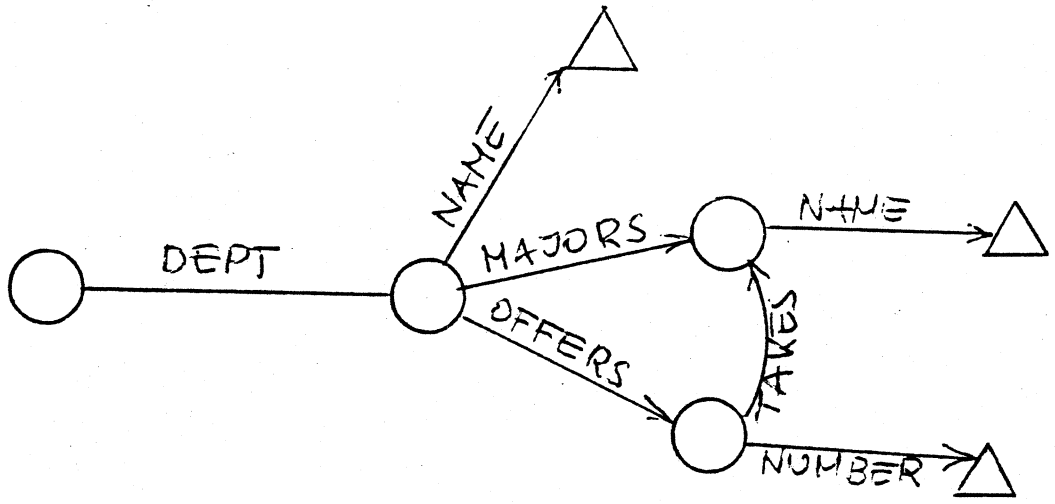


Fig. 2

The map is obtained by recursively collapsing all the arcs with the same attribute originating in a node into one, following by collapsing the corresponding end nodes of these arcs. The circle nodes of the map corresponds to sets of abstract records of GD while triangle nodes correspond to sets of data records.

Now we represent database of Fig. 1 as RD in the form of binary tables. We use the numbers attached to appropriate nodes as entries to the tables above while the heading of a table corresponds to a relations name (attribute).

DEPT	
	1
	2

MAJORS	
1	3
1	4
2	7
2	8

OFFERS	
1	5
1	6
2	9
2	10

TAKES	
3	6
4	5
4	6
7	6
7	9
8	6
8	9

NAME	
1	MATH
2	COMP
3	JOHN
4	LUCY
7	MARY
8	PAUL

NUMBER	
5	1003
6	2003
9	2003
10	3003

Fig. 3

It is easy to note that the graphical structure presented on Fig. 1 and even more clearly on Fig. 2 contains valuable information on efficient implementing the data base. It suggests keys, access structures and storage allocation. In contrast in the flat tables of the relational model described on Fig. 3 this potentially useful information is lost: it can only be recovered by converting the tables into a graphical or functional structure.

3. Combining logic programming with PROGRAPH.

In the following we shall present a proposal on how to deal with problems of logic programming mentioned in the Introduction. Our aproach will use the PROGRAPH data base model and programming technique. Our presentation will be based on examples.

Let us consider the following query:
"are all the students regular, where regular means faithful but not overzealous? A faithful student takes at least one course from the department in which he/she majors, while overzealous takes all the

courses from such a department".

Now we shall present the same query as a mixture of predicate logic and PROLOG:

(i) $\forall x$ **regular** (x)

(ii) **regular**(x) := **faithful**(x), **notoverzealous**(x)

(iii) **faithful**(x) := $\exists y \exists z[(DEPT(\text{令},y) \wedge MAJORS(y,x)) \supset (TAKES(x,z) \wedge OFFERS(y,z))]$

In the above query and following it definitions, let us distinguish two types of predicates: defined predicates like **regular**, **faithful** and **notoverzealous** and evaluation predicates, like DEPT, MAJORS, TAKES and OFFERS. The former we denote by using bold face letters while the latter by capitals. The defined predicates occur, at least once, on the left hand side of PROLOG expressions while evaluation predicates are attributes of the PROGRAPH data base or computation predicates like $x \geq y$ or $x+y=z$ (absent in our example).

Now we are ready to describe the processing of the query. We apply PROLOG mechanism to replace all occurences of the defined predicates, starting with the actual query in formula (i). (Let us note that the query is not negated or skolemized.)

These replacements will follow the rules of logic programming with the understanding that occurrences of universally qualified variables are treated as constants. Substituting (ii) into (i) with the appropriate unification, we obtain:

(v) $\forall x$(**faithful**(x) $\wedge$ **notoverzealous**(x)).

At that moment the comma ',' separating the two subgoals is converted into conjunction (' $\wedge$'). Now we continue our activity substituting into (v) the formulae (iii) and (iv) to obtain, after easy optimization:

(vi) $\forall x \exists y[(DEPT(\text{令},y) \wedge MAJORS(y,x)) \supset (\exists z(TAKES(x,z) \wedge OFFERS(y,z)) \wedge \exists u(TAKES(x,u) \wedge \neg OFFERS(y,u))]$

Let us note that (vi) does not contain any defined predicates and PROLOG phase of processing is therefore terminated.

In general the situation is more involved because in the presence of recursive definitions such a state cannot be achieved, but it is not a new phenomenon: PROLOG will deal with it in the same way as it usually does with recursion. It should be mentioned that the whole mechanism described above can be without the difficulties implemented in PROLOG.

Before we move to the next stage of producing a prograph corresponding to the formula (vi) let us provide some information about PROGRAPH.

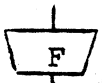
PROGRAPH is a programming functional language with a graphical front. It follows the direction of the Graphical Programming Language, GPL [2] developed at the University of Utah and dedicated to their data flow computer DDM 1 [1]. However, PROGRAPH goes much further then GPL allowing: compose operation, introduction of user defined subroutine, explicit indication of possible parallelism of computation and what is most important, it provides a mechanism for database access and update activities, which does not violate the functional character of the language. An experimental version of PROGRAPH is currently implemented on PERQ graphics station.

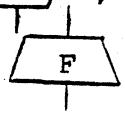
A PROGRAPH equivalent of 'program' is called 'prograph'. Generally speaking a prograph is a network of boxes connected by wires. A box, corresponds to a specific operation provided by the system (called primitive) or defined by a user. Such an operation is performed on datas supplied to its input and the results are delivered as outputs. The wires naturally connect inputs and outputs of distinct boxes without producing loops.

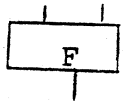
Now we shall introduce a few PROGRAPH primitives, necessary to present our query as a prograph.

Let F be an attribute of a PROGRAPH data base which we shall interpret, for the moment, as a binary relation. Let X be sets of nodes of data base applied respectively to input - top wire and Y resulting from output - bottom wire. As a matter of fact, inputs always are provided by top wires while outputs always are delivered by the bottom

ones.

The box 

, called access, means that Y= F[X] (in functional notation) while 

, inverse access, means $Y = F^{-1}[X]$. Now let x, y be single records provided as inputs of the box 

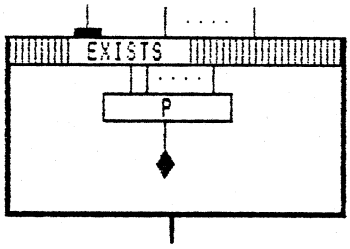
called application. Then the output z = F(x,y) (using relational notation). In this case the output is obviously of the type boolean, however PROGRAPH does not require specification of types of datas.

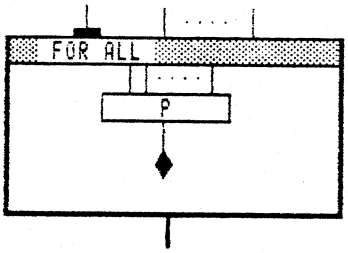
Let us introduce two obvious primitives: 

and 

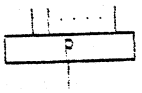
The oval boxes are used here only for visual effect so the user can easily distinguish logical operations.

Finally, we shall present two so-called composed operations (all the above ones are simple): EXISTS and FOR ALL.





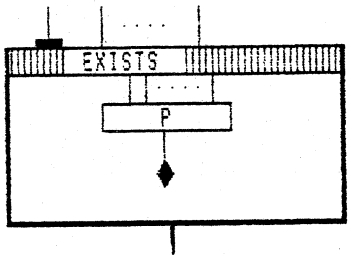
The number of inputs can vary while there must be one and only one called multiple which is signified by the  . It should be mentioned that multiple input does not have to be first to the left but it must not be more than one such input. It should be mentioned that the multiple input does not have to be first to the left.



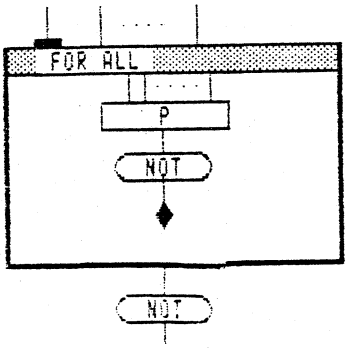
denotes an arbitrary prograph with k+1 inputs ($k \geq 0$) and one output of the type boolean. The semantics of these operations is: if $x, y, ,\ldots, y_k$ are values of inputs then the values of outputs are respectively:

$$\exists\, x(\, x \in X \wedge P(x, y, ,\ldots, y_k) \text{ and } \forall\, x(x \in X \supset P(x, y, ,\ldots y_k))$$

It is worth to note that the PROGRAPH definitions of quantifiers satisfy the basic properties of predicate logic: that is



is equivalent to



Now we are ready to present the PROGRAPH equivalent of the formula (vi)

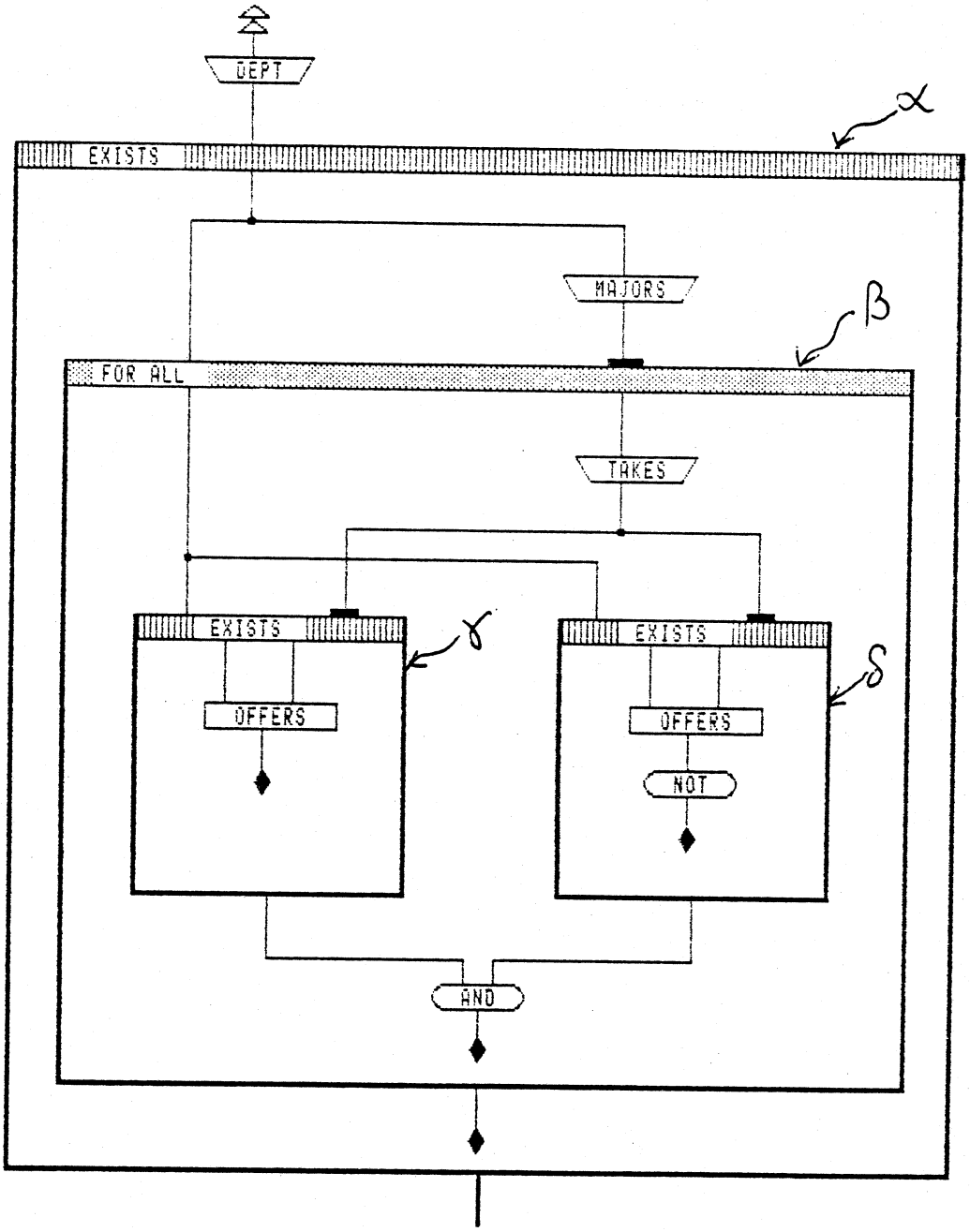


Fig. 4

Note that letters α, β, γ and δ are not part of the PROGRAPH description: they are introduced as references to appropriate EXISTS and FOR ALL boxes.

Now we shall present an informal description of how the prograph of Fig. 4 is derived.

First let us construct the following graphics presentation of the formula (VI) called outline which will be useful for our explanation.

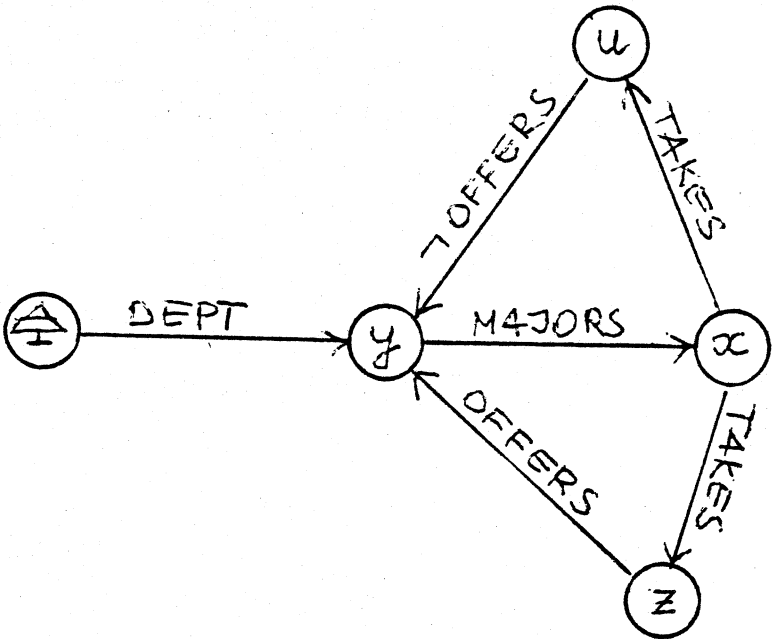


Fig. 5

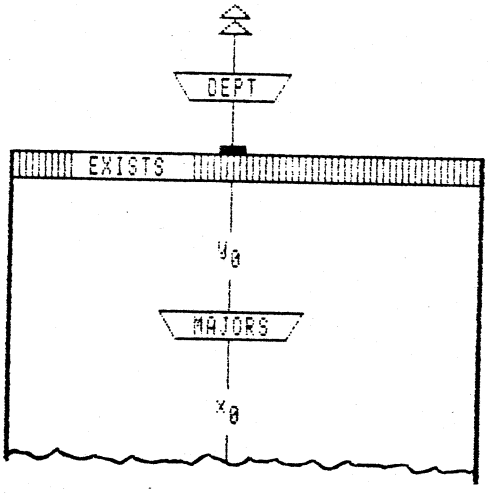
The outline is a directed graph with nodes corresponding to distinct variables in formula (VI). The labelled arcs correspond to predicates (or negated predicates) of (VI) in such a way that R labels arc originating in the node m and ending in n iff R(m,n) occurs in (VI). In order to derive the prograph of Fig. 4 we introduced a partial order among the arcs the outline which is imposed in natural way by the directions of arcs.

Now we can proceeded with constructing the prograph starting with a minimal arc (in this case: DEPT) and create:

Then we progress along outline and arrive at the end node of this arc (in this case node y), and create EXISTS box α since y is existentially quantified in (VI).

We proceed in an analogous manner with arc MAJORS: we introduce and create β box FOR ALL.

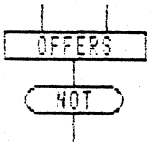
The rational behind the box is somewhat more complex so we will provide the reader with some additional explanation. Let us consider the top of the prograph of Fig. 5:



where y_o is the input to MAJOR and x_o the output. Obviously, y_o satisfies DEPT(, y_o) and $x \in x_o$ iff MAJORS(y_o,x). Therefore $x \in Xo$ iff DEPT(, y_o) $\wedge$ MAJORS(y_o,x) so in view of formula (vi) and definition of semantics of operation FOR ALL the introduction of the box is justified. Following the ordering of outline on Fig. 5 we arrive in node x and note that there are 2 arcs originating in x, both labelled TAKES. Therefore we introduce TAKES and branch the output. The corresponding branches are directed to EXISTS boxes and respectively, which corresponds to z and u, variables of Fig. 5. Now both arcs OFFERS and $\neg$ OFFERS end in the node y (already traversed). In this case we fill the boxes γ and δ with the

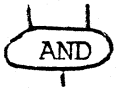
operations:

and



respectively

It is worth noticing that the first input wire corresponding to y variable originates in the box α, has been transmitted into β (first input wire) and branches there to arrive as first input to γ and δ respectively. Finally outputs of boxes γ and δ are joined as inputs to



box according to the conjunction of both existential subexpressions $\exists z(...)$ and $\exists u(...)$ in formula (VI).

The above description of an algorithm for producing a prograph from a well formed formula, as we mentioned already, is fairly infomal and sketchy. However, there is a formal algorithm performing this task which is unfortunately too lengthy to be described here and will be a subject of a separate publication.

To further convince the reader that the proposed approach is useful, we will present two more examples of data base queries and their PROGRAPH representation.

First query:

'does exist a course offered by the department of mathematics such that every student majoring in math takes this course'

Here is this query presented as a iff formula of predicate logic:

(viii) $\exists x \exists z \forall y[(DEPT(\text{⛪},x) \wedge NAME(x,MATH) \wedge MAJORS(x,y)) \supset (OFFERS(x,Z)\ TAKES(y,z))]$

Given below is an equivalent PROGRAPH formulation:

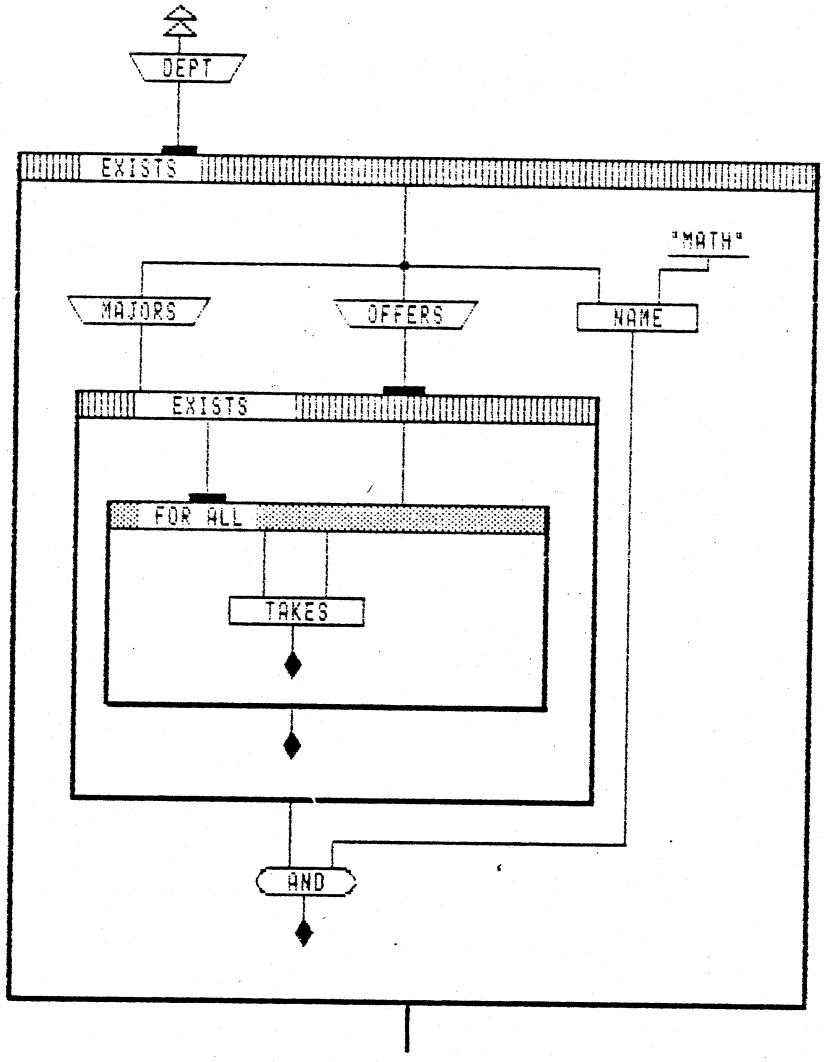


Fig. 6

Now we shall present a PROGRAPH version of the formula (viii) with a second and third quantifier reversed, so the prefix looks as follows: $\exists x \forall y \exists z$ and the matrix is unchanged.

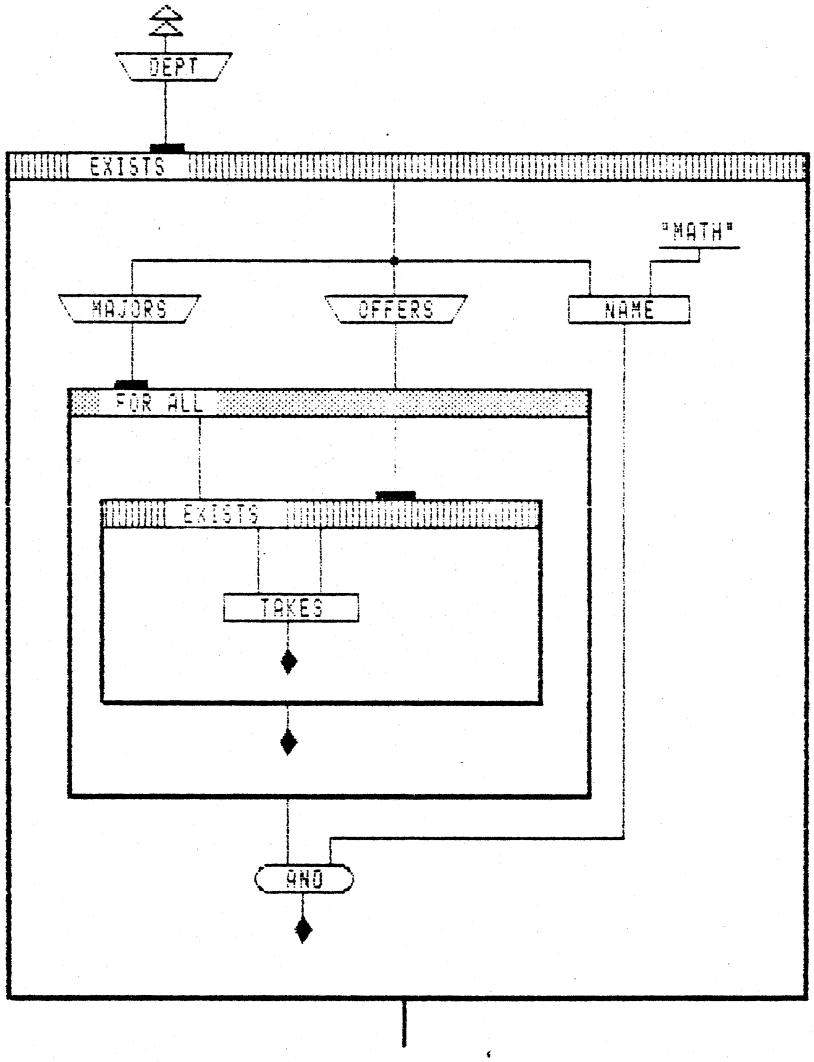


Fig. 7

The reader is encouraged, to find how the reversal of qualifiers effects the changes in corresponding prographs.

4. Conluding Remarks.

The presented results have a preliminary character, but in the author's opinion, leave no doubt that the approach is worth persuing. Since an experimental version of PROGRAPH is already functioning on a PERQ graphics station we intend to use it for a thorough series of experiments with a variety of data base queries formulated in terms of

PROGRAPH. The next stage will be to introduce an interface with a PROLOG implementation (unfortunately such one is not, at present, available on PERQ). This can be achieved by establishing appropriate communication with another computer or porting PROGRAPH onto a computer system where PROLOG is available. Finally, we would like to experiment with the combination of both as a uniform environment.

BIBLIOGRAPHY

[1] Davis, A. L., Keller, R. M., "Data Flow Program Graphs", IEEE Computer, Feb 1982, pp. 26-41.

[2] GPL Programming Manual, CS Dept., University of Utah, July, 1981.

[3] Pietrzykowski, T., "Programming Language PROGRAPH: Yet Another Application of Graphics" (with S. Matwin and T. Muldner), Graphics Interface 83 Conference, Edmonton, May 1983.

[4] Pietrzykowski, T., "Report on a Functional Language with a Graphical Front PROGRAPH: (with S. Matwin and T. Muldner), Research Notes, CS 83 02, School of Computer Science, Acadia University, 1983.

[5] Sato, T., "Negation and Semantics of PROLOG programs", Proceedings of 1st International Workshop on Logic Programming, Marseille, 1982.

[6] Van Emden, M. H., "Computation and Deductive Information Retrieval" in "Formal Description of Programming Concepts", North Holland, 1978.

[7] Warren, D., "Efficient Processing of Interactive Relational Data Base Queries Expressed in Logic", Proceedings of Conference on Very Large Data Basis, 1981.