Association for Computing Machinery

CSC '86

CINCINNATI

COMPUTER SCIENCE IN FOCUS: 1986

1986 ACM

Fourteenth Annual

COMPUTER SCIENCE CONFERENCE®

February 4-6, 1986

Cincinnati, Ohio

PROCEEDINGS

acm

A FORMALISM FOR VIEWS IN A LOGIC DATA BASE

José C.F.M. Neves*, George F. Lug er** and João M. Carvalho*

*Department of Computer Science
Minho University, Braga, Portugal

**Deparment of Computer Science
University of New Mexico, U.S.A.

Abstract

A formal specification of views for a logic data base is introduced based on an ordered set of first order logic languages. This formalism offers an efficient tool for the specification of a large class of data base operations. The data base query language Query-By-Example is extended to handle this class of data base operations. Finally, a formal characterization of the logic data base by the mathematical structure of a group is presented.

KEYWORDS: First Order Predicate Logic, Logic Data Base, Query-By-Example, Data Base View, Group, PROLOG.

1. Introduction

A data base of facts and general logic rules relating facts and other rules is a pre-requisite for natural language understanding, plan generation, and expert systems applications where an (often large) knowledge base is required. A formalism that approaches both data base and knowledge base management in an uniform manner, is reported in Neves (1). This formalism offers a predicate logic based specification and a practical language, PROLOG (Roussel (2), Warren (3)), for expert systems and data base design.

Query-By-Example (QBE, Zloof (4)) is a language developed for querying relational data bases (Codd (5)). The similarity between QBE syntax and PROLOG goals has been noted in the literature (Neves et al (6)).

This paper extends the logic data base system given in (1) to allow for data base views to be represented as logic rules. The data base query language QBE is extended in order to allow for the definition of this type of data base operation. Finally, the formal characterization of a logic data base system by the mathematical structure of a group is presented.

2. The Underlying Logic Data Base System

In a logic data base every relation, predicate, function or data base query must have a type associated with each of its argument positions. Each constant or variable must also be typed. The following conventions are used in this paper, viz.

- Variables are denoted by strings of symbols beginning with an upper-case letter or the symbol underscore "_".

- Constant are denoted by strings of symbols starting with lower-case letters.

- {X1,X2,...,Xn} denotes the set whose members are X1,X2,...,Xn.

- <X1,X2,...,Xn> denotes the ordered set or n-tuple whose members are X1,X2,...,Xn.

- An n-ary relation is a set of n-tuples.

- An n-ary relation is defined in a sentence of the form r(X1,X2,...,Xn) (e.g., r(X1, X2) for "X1 bears r to X2").

- With an n-ary relation r one associates the set of n-tuples that forms the extension of 'r', i.e., r = {<X1,X2,...,Xn> | r(X1,X2,...Xn)}. The symbol "|" is "such that".

- <X1/T1,X2/T2,...,Xn/Tn> ε r (r a relation), indicates that X1,X2,...,Xn are free variables of r and T1,T2,...,Tn are their types. <X1/T1,X2/T2,...Xn/Tn> is said to belong to the extension of relation r. "ε" is the element relation.

- An n-place predicate is interpreted as a set of ordered n-tuples; or if considered semantically, an n-place predicate is called a relation.

- By a function is meant the assignment (function) which binds an extension to each predicate constant in each interpretation.

- A term is defined inductively as:
(i) A variable is a term.
(ii) A constant is a term.
(iii) If f is an n-place function and R1,R2,...,Rn are terms, then f(R1,R2,...,Rn) is a term.

- An atom is a formula defined inductively as: if p is an n-place predicate and R1,R2,...,Rn are terms, then p(R1,R2,...,Rn) is an atom or atomic formula.

- A literal is an atom or the negation of an atom.

- A clause is a formula of the form: for all X1,X2,...,Xn (q V ¬ p1 ¬ p2 V ... V ¬ pn) where the q and each pi (1<=i<=n) is a literal and X1,X2,...,Xn are the variables ocurring in the q and each pi. This clause is denoted by q <- p1,p2,...,pn. The symbol "¬" is to be treated as "not", "V" is inclusive "or", and the reverse arrow "<-" is "if".

For background material on group theory, mathematical logic, and types see (7), and (8).

3. A Logic Data Base as a Group

Consider a department-store data base whose relations: "suppliers", "supplier-parts", "parts", "orders", and "manager" have types associated with each attribute position. In order to treat types as objects, a new relation "relation" is added to the data base, giving the type of the relations' types, viz.

{<suppliers, supplier-parts, parts, orders, manager> ε relation}
{<sno, sname, status, city> ε suppliers}
{<sno, pno, qty, deptno> ε supplier-parts}
{<pno, pname, color, weight> ε orders}
{<manager's name, deptno, address, age> ε manager}

The left-hand side of each sentence denotes the sequence of data types that are associated with the relation name on the right-hand side of the sentence. Thus, for the extension of each relation, and at the planning level, one has (Neves et al (9)):

{<suppliers/relation, supplier-parts/relation, parts/relation, orders/relation, manager/relation> ε relation| relation(suppliers, supplier-parts, parts, orders, manager)}
{<X1/sno, X2/sname, X3/status, X4/city > ε suppliers | suppliers(X1, X2, X3, X4) }
{< Y1/sno, Y2/pno, Y3/qty, Y4/deptno > ε supplier-parts | supplier-parts(Y1, Y2, Y3, Y4) }
{< Z1/pno, Z2/pname, Z3/color, Z4/weight > ε parts | parts(Z1, Z2, Z3, Z4) }
{< A1/pno, A2/deptno, A3/manager's name > ε orders | orders(A1, A2, A3) }
{< B1/manager's name, B2/deptno, B3/adress, B4/age > ε manager | manager(B1, B2, B3, B4) }

One may also have the general rule, viz.

{<Y1/sno,_,_,_> ε supplier-parts|for (each < Y1/sno, _,_,_> ε suppliers-parts holds (suppliers-parts (Y1,_,_,_) <- suppliers(Y1,_,_,_))) }

{suppliers(_,_,_X4) ε suppliers | for each< _,_,_, X4/city > ε suppliers holds (suppliers(_,_,_, X4)<-X4=london or X4=new york)))}

The underscore ("_") represents a unique variable existentially quantified.

The last two sentences are integrity constraints of type "value-based" (Williams et al (10)). The first of the sentences states that one can only have an entry for a given supplier in the "supplier-parts" relation, if an entry for the supplier concerned already exists in the extension of the "suppliers" relation. The second one states that only suppliers from London or New York can be suppliers to the department store.

At the domain level, the abstract specification of the department store data base above translates into the set of ground clauses and the general rules, viz.

suppliers(3, blake, 30, paris).
supplier-parts(2, 1, 75, 2).
parts (2, screw, red, 10).
orders(2, 1, peter).
manager(peter, 2, london, 5).
suppliers(1, smith, 20, vienna).
supplier-parts(1, 3, 25, 5).
parts(3, hammer, green, 80).
orders(3, 2, john).
manager(john, 5, paris, 45).

supplier-parts(Y1,_,_,_) -> true if all(<Y1/sno,_,_,_ > ε supplier-parts, (<-(value-based(<Y1/sno>, (supplier-parts (Y1,_,_,_) <- suppliers(Y1,_,_,_)),true))),true).

suppliers(_,_,_,X4) -> true if all (<_,_,_,X4/city > ε suppliers,(<-(value-based(<X4/city >,(suppliers(_,_,_, X4) <- X4=london;X4=new york), true))), true).

The arrow "->" is to be read as "evaluates to" and "all" is a logic function that calls for aggregation of the general form, viz.

holds(<Q1/T1,Q2/T2,...,Qj/Tj> ε U pr,(q<- p1,p2, ...,pi), S)<- p1,p2,...,pi.

This may be read as:

for all Q1,Q2,...,Qj in the domain of pr (r<=i) such that if each pr holds then bolds(<Q1/T1,Q2/T2,...,Qj/Tj> ε U pr, (q<- p1,p2,...,pi),S) holds.

Q1,Q2,...,Qj are objects (variable terms or constants) that may occur as arguments in q and each pi, and T1,T2,...,Tj are their types. S is the data base object that is returned as a consequence of the evaluation of the term conjunction p1,p2,...,pi. The symbol "U" models the process of set union.

Thus, the extensions of relations are tuples, the extension of integrity constraints are truth-values (i.e. "true", and "false"), and the extension of types are entities.

This, definition characterizes a logic data base system by the mathematical structure of a group. A logic data base is then given by the sequence, viz.

<e1,e2,...,en;r1,r2,...,rm;f1,f2,...,fk;o1,02,...,ol>

e1,e2,...,en are sets; r1,r2,...,rm are zero or more relations included in e1Xe2X,...Xen;f1,f2,...,fk are zero or more functions included in e1Xe2X...Xen; and o1,o2,...,ol are zero or more data base objects included in e1Ue2U...Uen (e1Xe2X...Xen denotes the cartesian product of e1,e2,...,en).

4. A QBE-Like Formulation of Views

A data view can be described either as a special kind of access constraint associated with one or more relations, as the extension of a data base relation (i.e., its current value), or as a relation containing only part of the information of a data base relation (or parts of several relations). This effectively shields data base information from the user(s). One must then be able, viz.

- To delegate one or more types of access rights to specific users
- To confine the user(s) to access only a subset of the data in the data base.

As an example, suppose that it is desired to set read-only access to relation "suppliers" for user Jones, for suppliers from London or Paris, who supply parts that are red. One may enter (the user's contribution is in bold):

suppliers	sno	sname	status	city
iv.|p.|.|jones|.	**X**			**D**

supplier parts	sno	pno	qty	deptno
	X	**Y**		

parts	pno	pname	color	weight
	Y		**red**	

CONDITIONS
D=london or D=paris

The entry in the tuple-command-field of the derived relation has the format, viz.

iv.|<user-operation>|.|<users-list>|. or
i.view.|<user-operation>|.|<users-list>|.

The entry |<user-operation>| is a list of one or more of the four rights "p." (print), "i." (insert), "u." (update) or "d." (delete). The entry |<user-list>| is the user(s) identification to the system. In generalizing this item, following the philosophy of QBE, variables may be used in place of <user-operation> or <users-list>.

As an example, suppose that one wishes to create a view "orders" that gives the items ordered for each department, with the manager ordering them. Access rights are to be granted only to the departments' managers. To fulfill this enter:

orders	pno	deptno	manager's name
iv.R.U.	A	B	U

supplier-parts	sno	pno	qty	deptno
		A		B

manager	manager's name	deptno	address	age
	U	B		

5. Compiling Views

The integration of information such as that found in the previous section into the knowledge base, requires the existence in the data base of a new relation, namely the "access-rights" relation with attribute-names: "user", "user-operation" and "relation-name". This is a ternary relation that is described in a sentence of the form: access-rights(S, T, U), where S is the user that is entitled to perform operation T on relation U. Formally:

{<user, user-operation, relation-name> ε access-right }
{<S/user, T/user-operation, U/relation-name> ε access-rights|access-rights(S, T, U)}

The first of the two QBE sentences of section 4 translates into the logic formulae (at the planning level):

{<X/sno, Y/sname, Z/status, C/city> ε suppliers|for (each<X/sno, Y/sname, Z/status, C/city> ε suppliers holds for(the <jones/user, p/user-operation, suppliers/relation-name> ε access-rights holds access-rights (jones, p, suppliers)) & for(each <X/sno, Y/sname, Z/status, D/city> ε suppliers holds (suppliers(X, Y, Z, D) & supplier-parts(X, A, B, C) & parts (A, E, F, G) & D=london; D=paris))}

{<jones/user, p/user-operation, suppliers/relation-name > ε access-rights|for(the <jones/user, p / user-operation, suppliers/relation-name> ε access-rights holds (access-rights(jones, p, suppliers))}

These sentences translate into a set of sentences of type "holds" to be added to the data base (at the methods level):

view(<<X/sno, Y/sname, Z/satus, D/city> ε suppliers, <jones/user, p/user-operation, suppliers/relation-name > ε access-rights>, (suppliers(X, Y, Z, D)), {< X, Y, Z, D >})<- the (<jones, p, suppliers > ε access-rights, (<- access-rights(jones, p, suppliers)), {<jones, p, suppliers>}), all(<X, Y, Z, D> ε suppliers, (< - suppliers (X, Y, Z, D), supplier-parts(X, A, B, C), parts(A, E, F, G), D=london; D=paris), {<X, Y, Z, D>})).

access-rights(jones, p, suppliers).

The sentences "view", and "the" are also instances of the general sentence "holds". "the" is a logic function that calls for aggregation; it returns the sole data base object that satisfies the term conjunction p1,p2,...,pi.

At the planning level, the last QBE sentence translates into the logic formulae, viz.

{<A/pno, B/deptno, U/manager's name> ε orders | for(the <U/user, R/user-operation, orders / relation-name> ε access-rights holds (access-rights (U, R, orders))) & for(each <A/pno, B/deptno, U/manager's name> ε orders holds (orders(A, B, U) & supplier-parts(_, A, B) & manager(U, B, _, _)))}

{<U/user, R/user-operation, orders/relation> ε access-rights|for(the<U/user, R/user-operation, orders/relation > ε acess-rights holds (acess-rights(U, R, orders))}

The following set of clauses is now added to the data base (at the methods level), viz.

view(<<A/pno, B/deptno, U/manager> ε orders, < U, R, orders> ε acess-rights>, (orders(A, B, U)), {<A, B, U>}) < <- the(<U, R, orders> ε acess-rights, (<- acess-rights(U, R, orders)), {<U, R, orders>}), all(<A, B, U> ε orders, (<- orders(A, B, U), supplier-parts(_, A, B), manager(U, B, _, _)), {<A, B, U>})).

acess-rights(U, R, orders).

Applying these rules to the data base results in the set of clauses that form a view; i.e. the views are managed by applying to the data base a set of logic rules that select from the clauses in the data base the ones that are the candidates to be used in a well defined data base operation by a specific user.

6. Translating View Updates on Data Base Relations

When updating a data base view, consistency must be maintained between the relation (or relations) being updated and those remaining in the data base. That is, when updating a data base relation, logic rules have to be triggered that will reflect the changes made on the view across all the data in the data base. This is accomplished through the use of logic rules called "triggers" (Neves and Santos (11)). As an example, consider the situation where the part-numbers for parts that are red are to be changed by a factor of 10. The data base operation is performed by a specific user (e.g., Jones below). To fulfill this Jones enters:

parts	pno	pname	color	weight
u.	Y	A	red	B
	X	A	red	B

CONDITIONS
Y=10*X

The entry "u." in the tuple-command-field of relation "parts" identifies the new entry in relation "parts" that will superseed (update) the old one. Formally:

{< Y/pno, A/pname, red/color, B/weight >:< X/pno, A/pname, red/color, B/weight> ε parts|for(each < Y/pno, A/pname, red/color, B/weight> ε parts holds (parts(Y, A, red, B)<- parts(X, A, red, B), Y=10*X))}

In an entry of the form M:N (with M and N are n-tuples), N refers to the tuple being updated and M refers to the tuple returned as a consequence of the updating operation. Thus, if "r" denotes the extension of relation "parts" back to the point in time previous to the update operation and "r" denotes its extension thereafter:

r'=r-{<X/pno, A/pname, red/color, B/weight> ε parts| parts (X, A, red, B) } U {< Y/pno, A/pname, red/color, B/weight> ε parts|parts(Y, A, red, B) & Y=10*X}

The symbol "-" models the process of set subtraction.

Note, however, that the "parts" relation shares common attributes with relations "suppliers-parts" and "orders" (namely the attribute name "part-number" (pno)). Any change to the attribute values of argument "pno" in the "parts" relation must therefore be reflected on the other two relations. This is necessary to reflect the effect of an update operation across the other relations in the data base that share common attributes with the relation being updated. This requires the existence of another relation, namely the "event" relation with attribute-names: "user", "user-operation", "relation-name", "old", and "new". This is a 5-ary relation that may be described by a sentence of the form, viz.

event(S, T, U, <O1, O2, ..., On>,<N1, N2, ..., Nn>)

In this sentence S is the user which performed the data base operation T on relation U changing the data base objects listed in sequence <O1, O2, ..., On> into the data base objects listed in sequence < N1, N2, ..., Nn >. Formally:

{<user, user-operation, relation-name, T1, T2, ..., Tn >ε event}

{<S/user, T/user-operation, U/relation-name, < O1/T1, O2/T2, ..., On/Tn>, <N1/T1, N2/T2, ..., Nn/Tn >> εevent| event(S, T, U, <O1, O2, ..., On>, <N1, N2, ..., Nn>)}

where <T1, T2, ..., Tn> ε U. Thus, for relation parts:

{<jones, u, parts, pno, pname, color, weight> ε event}

{<jones/user, u/user-operation, parts/relation-name, < X/pno, A/pname, red/color, B/weight>, <Y/pno,A/pname, red/color, B/weight>> ε event | event(jones, u, parts, < X, A, red, B>,<Y, A, red, B>)) & Y=10*X}

The predicate "event" is a data base relation. This is used to record changes in the attribute values of a particular relation that may interfere with attribute values in other relations, and the values of those changes. The extension of the "event" relation is built on the "update" operation performed on a particular relation. This implies that trigger-rules applied to the "supplier-parts" and "orders" relations must be entered into the data base. For the "orders" relation:

parts	pno	pname	color	weight
tevent.|u.|.L.	New			
	Old			

orders	pno	deptno	manager's name
i.taction.|u.|.	New	A	B
	Old	A	B

The entry "tevent.|u.|.L." in the tuple-command-field of the relation "parts" signals the events that set the update of the "orders" relation into action. The entry "i.taction.|u.|." in the tuple-command-field of relation "orders" signals the relation to be triggered and the kind of action set. Formally:

{<New/pno, A/deptno, B/manager's name >:< Old/pno, A/deptno, B/manager's name > ε orders | for(each <jones/user, u/user-operation, parts/relation-name, <Old/pno>, <New/pno>> ε event holds (event(jones, u, parts, <Old>, <New>)) & for(each<New/pno, A/depno, B/manager's name> ε orders holds(orders(New, A, B) < - orders(Old, A,))))}

The entry in the tuple-command-field of the trigger relation (i.e., the "parts" relation) has the format:

tevent.|<user-operation>|.|<users-list>|

The entry in the tupple-command-field of the triggered relation (i.e., the "orders" relation) has the format:

<user-operation>.taction.|<user-operation>|.

A similar rule applies to the "supplier-parts" relation. The concrete syntax of the full query language may be found in (11).

7. Conclusions

A formal characterization of a logic data base by the mathematical structure of a group has been presented. Expressing the view mechanism in QBE within the framework of the clausal form of predicate logic has been shown to be feasible and friendly from the user' point of view.

Relationships between the objects that make up the view are identified at several levels of abstraction in the view problem solving process. These are either from the problem solving task itself or the logic used to implement it. An advantage of the approach proposed here is that an input string in the form of a QBE sentence is easily scrutinized for purposes of manipulating it or generating PROLOG code.

Finally, the use of an ordered set of logic languages for handling data simplifies and clarifies some data base operations. These operations are translated into extended logic programs or goals that can be evaluated by using any standard PROLOG system.

Aknowledgements

This research is supported by North Atlantic Treaty Organization grant for International Collaboration in Research awarded to George F. Luger and José C.F.M. Neves.

References

(1) Neves, J.C.F.M., *The Application of Logic Programming to Data Bases*, PhD Thesis, Department of Computer Science, Heriot-Watt University, Edinburgh, Sctoland, November 1983.

(2) Roussel, P., *PROLOG - Manuel de Reference et d'Utilization*, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille, Luminy, France, 1975.

(3) Warren, D.H.D., *Implementing PROLOG - Compiling Predicate Logic Programs*, Research Reports 39 and 40, Department of Artificial Intelligence, University of Edinburgh, 1977.

(4) Zloof, M.M., *Query-By-Example - A Data Base Language*, IBM Systems Journal, 16, 4, 1977, 324-343.

(5) Codd, E.F., *A Relational Model for Large Shared Data Banks*, CACM 13, 6, 1970, 337-387.

(6) Neves, J.C.F.M., Anderson, S.O. and Willimans, M.H., *A PROLOG Implementation of Query-By-Example*, Proceedings of the 7th International Computing Symposium, March 22-24, 1983, Nurnberg, Germany, 318-332.

(7) Knuth, D.E., and Bendix, P.B., *Simple Word Problems in Universal Algebras*, in Computational

Problems in Abstract Algebra, J. Luch, (ed.), Pergamon Press Publishing Co., 1969, 263-297.

(8) Enderton, H.B., *A Mathematical Introduction to Logic*, Academic Press Publishing Co., New York, 1972.

(9) Neves, J.C.F.M., Luger, G.F., and Amaral, L., *Integrating a User's Knowledge into a Knowledge Base using a Logic Based Representation*, Research Report CS 85-2, The University of New Mexico, Department of Computer Science, Albuquerque, U.S.A., 1985.

(10) Williams, M.H., Neves, J.C.F. M., and Anderson, S.O., *Security and Integrity in Logic Data Bases using Query-By-Example*, Proceeding of the Logic Programming Workshop'83, Algarve, Portugal, 1983.

(11) Neves, J.C.F.M., and Santos, S.M., *Triggers in a Logic Data Base using Query-By-Example*, Proceedings of the Colloque International d'Intelligence Artificielle, Marseille, October 24-26, France, 1984.