A DEDUCTIVE QUESTION ANSWERING SYSTEM ON RELATIONAL DATA BASES

Koichi F'urukawa
Computer Science Division, Electrotechnical Laboratory
Tokyo, Japan

ABSTRACT

This paper describes a new formalization of a deductive question answering system on a relational data base using a theorem proving technique. A theorem proving procedure for a finite domain is investigated and a direct proof procedure based on substitutions of equivalent formulas which employs the breadth first search is intorudced. The search strategy is then expanded to set operations of the relational algebra which are incorporated into the proof procedure in order to increase the data base search efficiency. Virtual relations arc realized by means of introducing several axioms and utilizing the deductive capability of the logical system. Furthermore, a conditional domain is, introduced as one of the virtual domains and is used to give a relational view to a pseudo relational data base which can represent exceptional cases using some 1ink information.

A query transformation system called DBAP (Data Base Access Planner) which embodies those features is implemented in QJJSP.

1. Introduction

Many research groups in the artificial intelligence field have been concentrating their efforts on how to represent knowledge1 and how to perform logical inference and/or common sense reasoning. The knowledge data bases are organized in very complicated ways in order to realize those very high level functions. These structural and operational complexities have been preventing us from expanding them to very large knowledge data bases.

On the other hand, there have been many projects to develop very large commercial data bases in the data base research area. This kind of data base is assumed to be used in a relatively simple manner and consequently has simple structures:. Efficient search algorithms for such simple structures have been developed extensively and some special purpose hardware systems with parallel searching capability are being developed in many places.

Our current research goal is to combine these two separate efforts to build up a very large data base with the deductive capability [8], [II].

Codd, E. F. [2] introduced an algorithm to convert any query written in a relational sublanguage to a sequence of relational algebraic, operations in order to show the relational completeness of the relational algebra. His algorithm can be considered as a formal question answering (QA) procedure on a relational data base. On the other hand, Green, C. and Raphael, B. \k] formalized a deductive QA system based on first order logic. The essential point oV their formalism is that knowledge is represented by a set of axioms and the answer of the question is extracted from the refutation proof of that question.

In this, paper, these two formalisms are combined by introducing a proof procedure for a finite set, where logical expressions are interpreted as set operations on the set. A proof procedure for queries which require all answers satisfyingthegivenspecifico.tionispresented. It is a direct proof procedure based on substitutions of equivalent formulas. As an intermediate result of the direct proof, the system generates an access plan to the data base, and then the plan is executed to get the all answers satisfying the specification. The set operations of the relational algebra are considered as expanded notions of the breadth first search strategy and are incorporated into the proof procedure to express the accees plan.

Stonebraker, M. [10] introduced the notion of views (we call them virtual relations) in order to provide users with the deductive capability, and realized them by means of query modification. In this paper, virtual relations are considered to provide a semantic model of the base relations and are defined by a set of non-ground axioms. The query modification process can be considered as substitution process of a formula by an equivalent formula, the rule of which is given by the associated axiom. An axiom called a conditional domain axiom is particularly interesting. It is used to give a relational view to a pseudo relational data base which can represent exceptional cases using some link information.

In addition, some considerations on deletion of redundancies will be presented. Optimization of the access plan will also be eoneidered. The implemented query transformation system DBAP will be briefly explained. In the last section, the conclusion and some future research works to be done will be described.

2. Formalization

Generally, a formal QA system consists of a set of axioms and a theorem prover to get answers for a given query. Fig. 1 shows the configuration of our system in terms of the formalism.

In a formal system, each datum in the data base has to be expressed by a ground clause (a clause which does not contain any variables). There are two typical representations: namely, the tuple-wise representation and the domain-wise

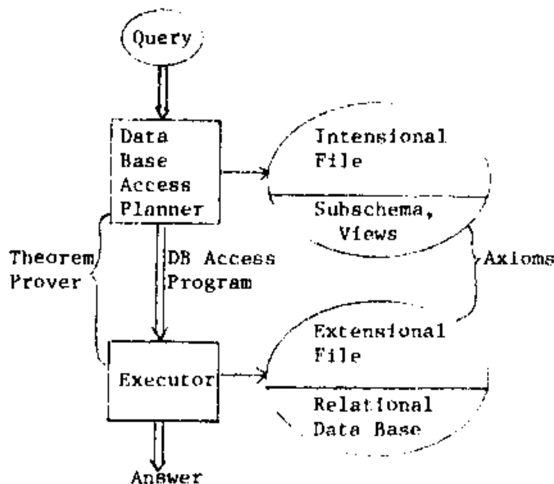


Fig. 1. A formal QA system on a relational data base.

```
EMP(NAME  DNAME  SAL)

    SMITH  R&D    12000
    BROWN  SALES  16000
           . . .
```

(a)

```
EMP.NAME(1, SMITH)
EMP.DNAME(1, R&D)
EMP.SAL(1, 12000)
EMP.NAME(2, BROWN)
EMP.DNAME(2, SALES)
EMP.SAL(2, 16000)
     . . .
```

(b)

Fig. 2. An employee relation EMP and its ground clause representation.

representation. Each ground clause of the tuple-wise representation expresses a tuple of some relation, whose name is designated by the name of the predicate. If we use this predicate to express queries, the expression looks like DEDUCE expression [1] which is considered as one of the relational algebraic sublanguages. On the other hand, the domain-wise representation is related to the relational calculus. Since the relational calculus is closely related to the general predicate calculus, the domain-wise representation is expected to suit the formal system more naturally than the tuple-wise representation. This is why we adopt the domain-wise representation.

In order to designate a datum in a relational data base, we need to specify a relation name, a domain name and a tuple identifier. We introduce a 2-place predicate <relation name>·<domain name> which has a tuple identifier (tuple id) as its first argument and the datum as its second argument. A tuple id associates clauses which constitute the tuple. An example relation EMP is shown in Fig. 2(a) and the corresponding ground clause representation is shown in Fig. 2(b).

Green, C. [3] introduced the ANS predicate to express a query by a logical formula. The query "Get the names of all employees who belong to the research and development (R&D) department" is expressed logically by

$$(\forall x)(\forall i)(\mathrm{EMP.NAME}(i,x) \wedge \mathrm{EMP.DNAME}(i,\text{'R\&D'}) \rightarrow \mathrm{ANS}(X)).$$

This formula means that every name of the employees who belong to the 'R&D' department is an answer, but does not mean that all such names are required.

We adopt the following representation:

$$(\exists ?x)(\exists i)(\mathrm{EMP.NAME}(i,?x) \wedge \mathrm{EMP.DNAME}(i,\text{'R\&D'})).$$

That is, we express the required variables by putting a prefix symbol and read the expression as 'Find ail ?x such that (i)...'.

The intentional file consists of non-ground axioms which define users' views or virtual relations. The objective of introducing users' views is to keep the query language independent of the logical structure of the relational data base.

Assume that we have a relational data hase which consists of the following base relations:

EMP(NAME, DNAME SAL)

DEPT(WAME MGR)

where the domais DNAME in EMP and NAME in DEPT are both the set of departments. Assume also that a user wants to define a virtual real!ion VEMP(NAME DNAME SAL MGR). In the virtual relation VEMP, the domain MGR belongs to the employee relation, but in fact it belongs to the department relation. The domain MGR is considered to have been transfered from the department relation to the employee relation, and we call this kind of virtual domain <u>a transitive domain</u>.

in terms of the VEMP relation, the fact that the manager of an employee i is x is expressed as VEMP.MGR(i,x). The QA system has to transform this expression to the following conjunction of literals on the base relations:

$$(\exists j)(\exists y)(\mathrm{EMP.DNAME}(i,y) \wedge \mathrm{DEPT.NAME}(j,y) \wedge \mathrm{DEPT.MGR}(j,x)).$$

This formula is deduced by applying the following equivalence statement on the former expression:

$$(\forall i)(\forall x)(\exists j)(\exists y)(\mathrm{VEMP.MGR}(i,x) \leftrightarrow \mathrm{EMP.DNAME}(i,y) \wedge \mathrm{DEPT.NAME}(j,y) \wedge \mathrm{DEPT.MGR}(j,x)). \quad (1)$$

This type of statement is called <u>a transitive axiom</u>.

Let us consider the query "Who is the manager of Mr. SMITH ?". In terms of the virtual relation VEMP, this question is logically expressed by

$(^3?x)(^3i)$(VEMP.NAME(i, 'SMITH'
A VEMP.MGR(i,?x)). (2)

By substituting the second term in (2) by the righthand expression of the equivalence sign = in (l), we obtain the following expression:

$(^a?x)$(3i)(aj)(3y)
(VEMP.NAME(i,'SMITH') A EMP.DMME(i,y)
A DEPT.NAME(j,y) A DEFT.MGR(,1,?x)).

So for, we have obtained the expression in terms of the base relations except the underlined literal. This literal is transformed to the corresponding base relation literal by the following axiom:

$(^Vi)(^Vx)$(VEMP.NAME(i,x) i EMP.NAME(i,x)).

This type of axiom is called a simple domain axiom, and a query which does not include any virtual relation literals is called a base query.

It is obvious that any query which is specified in terras of virtual relations is translated to an equivalent base query by logical inference. However, the trnasfromation by the resolution rule which is based on the modus ponens is insufficient if we want to get all answers which satisfy the given specification. We can prove it easily. Denote a query by F[?x] and the required answers by {?x| F[?x]}. If we obtain a base query G[YxJ by applying the resolution rules, then G[?x] ~v F[?x]. Therefore, {?x| G[?xIl £ {?x| Ff?x]l, where the equality holds only if G[?x] ^ F[VxJ.

So far, these transformations can be realized by the query modification technique [10]. As far as the control structure is concerned, it is equivalent to the input resolution in the GL-resolution which is known to be valid only for a horn set [13]. But there exist more complicated axioms which require the whole inference capability including the ancester resolution. We will introduce a few such axioms later on.

A virtual domain can be defined in terms *of* other predefined virtual domains. The axioms for such domains transform a literal to a conjunction of literals some of which are not the base re1ation 1iterals.

In this paper, we consider only existentially quantified queries. It is easily shown that the resulting base queries after applying the transformations are also only exsistentially quantified. Therefore, we further simplify the notation for queries by omitting all quantifiers.

3. Deletion of Redundancies

A base query which is obtained so far may have some redundancies. Let us consider the base relations: EMP(NAME DNAME); DEPT(NAME MGR LOG), and the virtual relations: VEMP(NAME DNAME LOG); VDEPT(NAME MGR). Note that the LOG domain is transitive. Assume that the following query is given:

VDEPT.MGR(j,?x) A VEMP.L0G(i,1*1*2)
A VEMP.DNAME(i,y) A VDEPT.NAME(j,y) (3)

Tt inquires the manager of the department y to which the employee i located at 1*1*2 belongs. The expression (3) is transformed to the following base query:

DEPT.MGR(,1,?x) (M
A EMP.DNAME(i,z) (5)
A DEPT.DNAME(k,z) (6)
A DEPT.LOC(k,1*1*2) (7)
A EMP.DNAME(i,y) (8)
A DEPT.NAME(t1,y). (9)

Let us concentrate our attention on the literals (S) and (8). They are the same except the values. However, as a datum is uniquely designated by specifying the relation name, the domain name and the tuple id, these two variables z and y must refer the same datum. This fact can be expressed by the following axiom:

Tuple Id Axiom

For any relation REL and its arbitrary domain D, the following statement holds:

$(^Vi)(^Vx)(^Vy)$(REL.D(i,x) A REL.D(i,y)
= REL.D(i,x) A x = y). (10)

By applying this axiom, to the above two literals, y is replaced by z and these two literals become exactly the same. Further, the literal (9) is transformed to

DEPT.NAME(j,z). (9)'

This simplification is not the same as the factoring operation in the resolution proof procedure, because all variables are quantified by a. Therefore, the tuple id axiom is a meaningful axiom. Note that the application of this axiom cannot be done by input resolution, because we need to resolve two literals simultaneously.

Now, let us assume that the domain NAME in the DEPT relation is a key domain. Then, it is easily shown that the literals (6) and (9)' are the same. This fact is represented by the following axiom:

Key Domain Axiom

For any relation REL and its key domain K, the following statement holds:

$(^Vi)(^Vj)(^Vx)$(REL.K(i,x) A KEL.K(j,x)
= REL.K(i,x) A i = j). (U)

By applying this axiom on (6) and (9)', we obtain the following simplified base query:

DEPT.MGR(k,?x) A EMP.DNAME(i,z)
A DEPT.NAME(k,z) A DEPT.L0C(k,1*1*2). *(I?)*

1*. Data Base Accesses and Proof Procedure

When a proof contains multiple data base accesses, it is recommended to plan the data base accesses very carefully in order to keep the proof process efficient. We will set up the following design objectives:

1. Do not access to the (logically) same tuple in a relation more than once.
2. Get all tuples which satisfy the given conditions to a certain relation at a time.
3. When more than one tuple are to be accessed, plan the access order to minimize the number of data base accesses.

As mentioned in section 2, each constituent of each tuple is expressed by a ground clause. But the expression is merely conceptual and we do not have such representations in the actual data bases. Instead, relational databases are usually organized in such a way that the data base access by tuples is much more efficient than by domains. In order to achieve the tuple-wise access to the data base, all literals in a base query associated to the same tuple of the same relation have to be grouped together. We call such a set of literals an access subclause. Since all literals in an access subclause have the same relation name and the same tuple id which is introduced only to associate those clauses in the same tuple, we can abbreviate the notation for access subclauses by factoring out the relation name and omitting the tuple id. For example, the access subclauses of (.1 ?) ar e expr es sed as.:

AI: DERI[1] (MGR(?x), NAME(z), L0C(M2)J, (13)
A2: EMP {DNAME(z)}. *(lh)*

The second objective is deeply related to the discussion on the inference rule mentioned in section ?. The same argument holds in getting data; namely, any access subclause must be substituted by an equivalent set of tuples. But since the set consists of all tuples which satisfy the access subclause, it is obtained by the associative retrieval with the breadth first-search strategy.

The breadth first associative retrieval operation (we denote it by *r)* on an access subclause can be expressed by a compound operation of the selection and the projection *of* the relational algebra. Denote the selection of a relation REL1 with a condition Di = a by RELI[Di = a] and the projection of a relation REL? to the domains Dj ,. ..,Dk as REL2(Dj . . . Dk). Assume that

A = REL DI)I(xI),... J)i(xi) ,Di+I(c)} (15)

where x],...,xi are variables and c is a constant. Then, r(A) is given by the following algebraic expression:

r(A) - REL[Di+I = C](D1 . . . Di.) . (I6)

For example, the application r on (13) results in:

r(AI) ~ DEPT[LOC = 442MGR NAME). (17)

We denote the relation resulting from the application of r on A by A..

Now, let us consider the third design objective. Any query can be described in PLANNER language [6] simply by expressing each access subclause by a goal statement with a corresponding associative retrieval pattern. But as mentioned earlier, the depth first search strategy employed in it is very inefficient when we want to get all answers which satisfy the given condition. On the other hand, CONNIVER [12] has a programming support to deal with the breadth first search strategy, but programmers are responsible for controlling the overall proof procedure. We will generate an efficient data base access program from the given set of access subclauses. This approach resembles the PODB's approach developed by Haral.dson, A. [s].

Let us consider the case in which there are more than one access subclauses. We say two access subclauses are associated if and only if they share at least one variable in common. Assume that two associated access subclauses are given. The proof procedure first obtains two separate one-level search trees AI and A2 by executing each associative retrieval. Then, it generates another one-level tree which consists of all answers satisfying both access subclauses by equating the shared variables in these trees. This operation corresponds to the equi-Join operation of the relational algebra [I|, [2]. We denote the equi-join of AI and A? with common variables xI,...,xi by AI[xI ... xi]A2, or simply by A1.A2 when the common variables are not required to be specified explicitly.

The association relation is a binary relation and can be described by a graph having access subclauses as nodes and the shared variables on the corresponding arcs. We call this graph an association graph. When there are three access subclauses, the corresponding association graph is either straight-line as shown in Fig. 3 or triangular as shown in Fig. 4. In either cases, the result is obtained by executing two successive join operations in an arbitrary order. In order to specify the order of join operations and give proper output relations, we introduce a kind of tree called a program tree. It is constructed from the association graph by an algorithm P which will be given in the appendix. Some examples shown in Fig. 5 and 6 may be helpful to get the idea of program trees. The program trees in Fig. 5(a) - 5(c) correspond to the association graph in Fig. 3 and Fig. 6(a) - 6(c) to Fig. 4. Let us consider about the execution of a program tree. A leaf node is executed at first, and a father node is executed only after all of its sons are executed, where an execution of a node A consists of r(A), followed by the join operations with its all sons (no join operations are defined for leaf nodes). The multiple join operations for a single node can be done in an arbitrary order. Moreover, they can theoretically be done simultaneously. The multiple joins for a branched tree TI in Fig. 7 are expressed by

Bn.(Bn-I.(.. .(B2.(B1.A))...)).

On the other hand, the successive joins for a straight-lined program tree T2 in the same figure are expressed by

Bn.Bn-1____B2.B1.A.

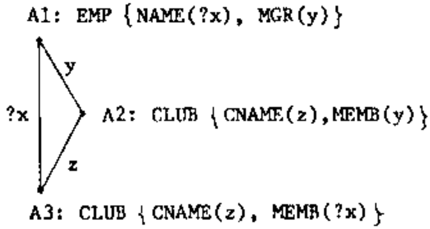


Fig. 3. An association graph which contains a circuit.

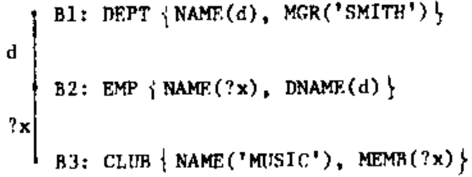


Fig. 4. An association graph which does not contain circuits.

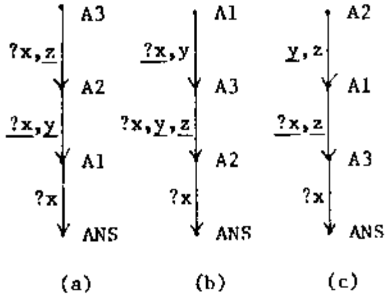


Fig. 5. Three program trees of the association graph in Fig. 3.

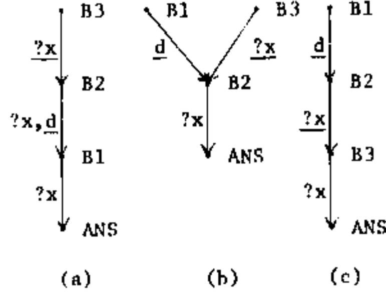


Fig. 6. Three program trees of the association graph in Fig. 4.

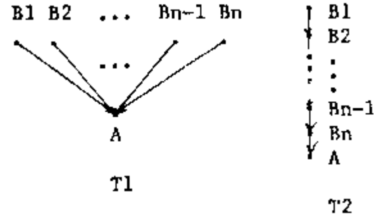


Fig. 7. Two extreme types of program trees.

Note that these expressions are evaluated from left to right. The new relation obtained by executing a node consists of only the domains which are to be used in the succeeding operations. We will put the variables which represent the output relation on the arcs from the node to the father node. Some of them are shared by the father node and the groundfather node, which in turn will be used as the link variables on the corresponding Join operation. In Fig. 5 and 6, the underlined variables are of this kind. As a result, the algebraic expression for a program tree, e.g. Fig. 6(a), is given by:

$$ANS = (((\underline{B3}\ [?x]\ \underline{B2})(?x\ d))\ [d]\ \underline{B1})(?x). \quad (18)$$

A base query may contain some comparison literals between two variables and/or between a variable and a constant, such as $y = z$ or $y > 15000$. They are put on the appropriate nodes or arcs in the association graph.

Now, the third design objective is restated as the problem of constructing an optimal program tree. It is reasonable to measure the efficiency of a program tree by the total size of all output relations generated during the proof process. The size of a relation is proportional to both the domain number and the tuple number in it. The optimization of the program tree construction algorithm is done by embedding a few heuristic strategies which select a suitable node in the association graph. The algorithm P constructs a program tree in reverse order to its execution. Therefore, we select less restricted nodes earlier. The heuristics we adopt are the following:

1. Select a node with smaller degree earlier.
2. Select a node with less constant literals earlier.
3. Select a node with more ? variables earlier.

The heuristics 1 is applied prior to 2 and 2 prior to 3. We denote the algorithm P with these heuristics as P*.

5.. Conditional Domain

It is not easy to deal with exceptional cases in the relational data base. For example, assume that the domain NAME in Fig. 2 is a key. Since no employees are allowed to appear in more than one tuple, this organization is adequate only if no employees belong to more than one department. However, it may happen that, say, Mr. SMITH has come to belong to both 'R&D' and 'SALES'. Then, we cannot express this fact in this relation as

long as we keep the NAME domain as a key domain. The traditional way to manage this situation is to use a general schema to represent many-to-many correspondence. That is, the domain DNAME is removed from the relation EMP and a new relation ED (NAME DNAME) is created to store all correspondences between employee names and department names, as shown in Fig. 8. This inconvenience is due to a strong constraint on a relational data base which requires that all data in a domain must be homogeneous. It is more natural to treat the exceptional cases as exceptions. We use a special symbol, say '*', to represent the exceptions. In the above example, an '*' is put on the Mr. SMITH'S DNAME field and only the two pairs, <'SMITH' 'R&D'> and <'SMITH' 'SALES'>, are stored in the newly created ED relation, as shown in Fig. 9- The domain DNAME is no more homogeneous, because the special symbol '*' does not belong to the domain of department names. This symbol can be considered to carry link information to the ED relation.

It is desirable to protect users from the struictural change of the data base by supplying the old relation EMP(NAME DNAME SAL) as a virtual relation. To avoid the conflict of the relation names, we rename the base relation EMP as, say, CEMP.

The fact that the employee i's department is x is expressed as EMP.DNAME(i,x), but the actual information is not always in the CEMP relation. In some case, it is in the ED relation. Therefore, we require a conditional treatment. We call this kind of domain in the virtual relations <u>a conditional domain</u>. A conditional domain is defined in terms of a conditional statement as follows:

$$(\forall i)(\forall x)(\exists z)(\exists u)(\exists j)(\text{EMP.DNAME}(i,x) \equiv (\text{CEMP.DNAME}(i,z) \wedge \text{CEMP.NAME}(i,u) \wedge (\underline{\text{IF}}\ z = \text{'*'}\ \underline{\text{THEN}}\ \text{ED.NAME}(j,u) \wedge \text{ED.DNAME}(j,x)\ \underline{\text{ELSE}}\ z = x)). \quad (19)$$

This axiom is called <u>a conditional domain axiom</u>.

Now, we will consider how to deal with the conditional expression. Assume that the following query is given:

$$\text{EMP.NAME}(i,?x) \wedge \text{EMP.DNAME}(i,\text{'SALES'}). \quad (20)$$

This query can be transformed to

$$\text{CEMP.NAME}(i,?x) \wedge \text{CEMP}(i,z) \wedge (\underline{\text{IF}}\ z = \text{'*'}\ \underline{\text{THEN}}\ \text{ED.NAME}(j,?x) \wedge \text{ED.DNAME}(j,\text{'SALES'})\ \underline{\text{ELSE}}\ z = \text{'SALES'}). \quad (21)$$

by using the conditional domain axiom (19), a simple domain axiom for the simple domain NAME and the tuple id axiom.

Let us denote the conditional expression in (21) as B. The literals in each branch of B are then grouped separately in order to make access subclauses. Then, if there are literals outside B which are to be contained in any access subclauses in B, they are distributed to every branch of B and put into the corresponding access subclauses. After that, the rest literals are also grouped to make the access subclauses. The result of applying these steps to (21) are given as follows:

$$\text{CEMP}\ \{\text{NAME}(?x),\ \text{DNAME}(z)\} \wedge (\underline{\text{IF}}\ z = \text{'*'}\ \underline{\text{THEN}}\ \text{ED}\ \{\text{NAME}(?x),\ \text{DNAME}(\text{'SALES'})\}\ \underline{\text{ELSE}}\ z = \text{'SALES'}). \quad (22)$$

Generally, the result consists of one or more access subclauses and one or more conditional expressions. We regard this kind of conditional expression as an access subclause and call it a_ <u>conditional access subclause</u>. A program tree is constructed by applying the algorithm P* introduced in section 4. The program tree of (22) is shown in Fig. 10.

Now, we will consider the execution of the program tree. Generally, a conditional access subclause C has the following form:

$$(\underline{\text{IF}}\ P\ \underline{\text{THEN}}\ C1\ \underline{\text{ELSE}}\ C2)$$

where CI and C2 are sets of access subclauses

CEMP(NAME	SAL)
SMITH	12000
BROWN	16000
. . .	

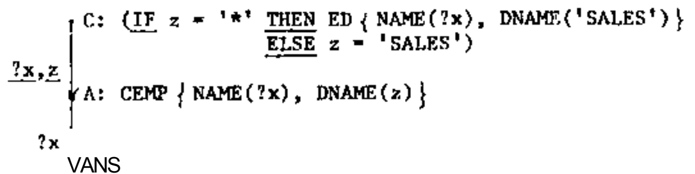
ED(NAME	DNAME)
SMITH	R&D
SMITH	SALES
BROWN	SALES
. . .	

Fig. 8. A reorganized relational data base to express an exception.

CEMP(NAME	DEPT	SAL)
SMITH	*	12000
BROWN	SALES	16000

ED(NAME	DNAME)
SMITH	R&D
SMITH	SALES

Fig. 9. A more natural way to express the exception.



VANS

Fig 10. The program tree of (22).

♦ A: CEMP) NAME ('SMITH[1]), DNAME (z) \

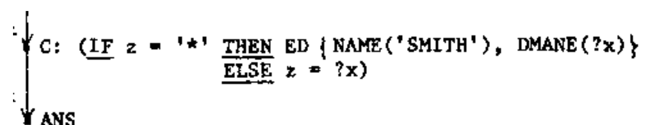


Fig 11. The program tree of (23).

and/or conditional literals. This expression is equivalent to the following OR expression:

$$(P \wedge C1) \vee (\sim P \wedge C2).$$

Assume that the output relation A of C's father A contains all variables which appear in P. Then, the conditional join between C and A can be defined by:

$$\underline{C1}.(\underline{A}\,[P]) \cup \underline{C2}.(A\,[\sim P]).$$

The algebraic expression for the program tree in Fig. 10 is given by:

$$ANS \equiv (\underline{C1}\,[?x]\,(\underline{A}\,[z = '*'])) \cup (\underline{C2}\,[z]\,(\underline{A}\,[z \neq '*']))$$

Note that the literal P and HP are passed to the father node A and tneir evaluations are delayed until the node A is evaluated.

As a matter of fact, the literal P (and ~P) can often be evaluated at the evaluation time of C. Consider the following query:

$$EMP.NAME(i,'SMITH') \wedge EMP.DNAME(i,?x). \quad (23)$$

The corresponding program tree shown in the Fig. 11 has such a property. Tn this case, we need not postpone the evaluation of P. Furthermore, it is evaluated in conjunction with A. Namely, the relation .A can be divided into two subrelatio $\underline{Ap}$ and $\underline{A{\sim}p}$ such that a tuple satisfies P if and only if it belongs to $\underline{Ap}$. The conditional Join between A and C is expressed by

$$(\underline{A}\,[P]).\underline{C1} \cup (\underline{A}\,[\sim P]).\underline{C2}.$$

In this case, the literal P works as a conditional branching statement for the node C if A_ consists of only one tuple. Therefore we express the conditional access literal 0 by the following COND statement:

(COND (P C1)(T C2)).

We can choose the OB expression or the COND expression properly in the construction time by investigating the variables on the arcs between the conditional access literal node and its sons.

Another approach to deal with the conditional case might be to transform a query to a disjunctive normal form and to solve each conjunction separately. But it is difficult to remove the redundancies caused by the separation. This is the reason why we keep the conditional statement in a unit form.

6. The Implementation of DBAP

The upper half of the total system shown in Fig. 1 was implemented on an AI language called QLISP [9]. The DBAP is not a formal theorem prover like the resolution theorem prover. The various kinds of heuristic strategies described through section 2 - 5 were realized by informal procedural methods.

Virtual domain axioms are actually QLAMBDA functions which are invoked by the patterns of virtual literals in a query, and perform the corresponding transformations. The axiom definer was implemented in order to define virtual domain axioms through their logical expressions.

The tuple id axioms are treated in a very different way. There are no explicit functions for the tuple id axioms. The deletion of redundancies is done by a search and substitution procedure embedded in the DBAP.

The key domain axioms are defined in a simpler form than (11); for example,

$$(\forall i)(\forall x)(DEPT.NAME(i,x) \equiv DEPT.NAME(x,x) \wedge i = x).$$

This axiom replaces the tuple id i by the value x and causes the same effect as applying the original key axiom (11). This treatment of key domain axioms solves the interaction problem between tuple id axioms and key domain axioms, and therefore increase the efficiency.

7- Conclusion

This research is considered to be a step toward a natural language QA system. In order to access the data base through the user's intention, the semantics or the real world model of the data base must be represented explicitly and be used to remove the gap between the semantic expression of a query and the logical data base structure.

On the other hand, the virtual domain axioms can be considered to be a representation of the data base semantics, because the virtual relations which are defined by a user can be regarded as his conceptual model of the real world [11].

We limited our consideration on queries modified only by the existential quantifiers. Universally quantified queries are related to the division operation of the relational algebra as shown by Codd, E. F. *[2].* It is expected that we can deduce the division operation if we pose the range separability condition [2] on the queries.

Another difficult problem occurs if some of the virtual domains are associated more than one conjunction of base relation literals. In this case, the whole virtual domain axioms are expressed in an and-or graph. Therefore, a general algorithm which performs the breadth first search on an and-or graph will be required.

Acknowledgement

This research was mainly done during my stay at SRI. I would like to thank Dr. Bertram Raphael and other many people who helped me to study there. I would like to especially thank Dr. Daniel Sagalowicz and Dr. Earl Sacerdoti for their stimulations and important suggestions concerning the ideas in this paper.

References

Chang, C. L., "DEDUCE - A deductive query language for relational data bases," To appear in Artificial Inteligence and Pattern Recognition, (ed. Chen, C. H.), Academic Press.

Codd, E. F., "Relational Completeness of Data Rase Sublanguages," Courant Computer Science Symposium 6, Prentice-Hall, (197?)•

Green, C, "Application of theorem proving to problem solving" Proc. 1st 1 JCAI, pp.219-239, (1969).

Green, C, and Raphael, B., "The use of theorem proving techniques in question answering systems," Proc. 23rd Nat. Conf. ACM, Brandon Press, Princeton, New Jersey, (1968).

Haraldson, A., "A procedure generator for a predicate calculus data base," Proc. IFTP Congress 74, pp.575-579, (1974).

Hewitt, C, "Description and theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot," AI Memo No. 251, MIT[1] Project MAC, (April 197?).

Levien, R. E., and Maron, M. E., "A Computer system for inference execution and data retrieval," CACM Vol. 10, No. 11, pp.715-721, (November, 1967)•

Raphael, B., The Thinking Computer, W. H. Freeman and Company, San Fransisco, (1976)•

Sacerdoti, E. D. et. al., "QLISP - A language for the interactive development of complex systems," Proc. AFIPS 1976 NCC, Vol. 45, pp.349-356, (1976).

Stonebraker, M., "implementation of integrity constraint and views by query modification," Proc. 1975 SIGM0D Workshop on Management of Data, San Jose, Calif., pp.65-78,(May, 1975).

Smith, J.M. and Smith, D. C. P., "Data abstruetion: Aggregation and generalization," To apear in ACM Transactions on Database Systems.

Sussman, G. J. and McDermott, D. V., "From PLANNER to C0NN1VER - A genetic approach," Proc. FJCC, pp.1171-1179, (197?).

VanderBrug, G. and Minker, J., "State-Space, Problem Reduction, and Theorem Proving - Some Relationship," CACM Vol. 18, No. 2, pp.107-115, (February 1975).

Appendix

Algorithm P. [Construction of a program tree] This algorithm constructs a program tree in the reverse order to the execution through marking the nodes and arcs of the association graph. We use M and N to denote nodes in the association graph, and use Mt and Nt to denote the corresponding nodes in the program tree respectively.

1. [Root construction]
   Put a node ANS as the root of the program tree. Then, select an arbitrary node N in the association graph and put it above the root as a son node. Mark the node N.
2. [Tree construction]
   If there is an unmarked arc from the node N (we denote this arc as A), then mark it and move to the other node M of arc A.
      Tf the node M is, marked, then put the variables of arc A on every arc which is on the pass from the node Nt to the node Mt. Go to 2.
      Otherwise (i.e. if the node M is unmarked), put the node M as a son of the node Nt and put the variables of arc A on the new are. Mark the node M. Rename the node M as N. Go to 2.
   Otherwose (i.e. if all arcs from the node N are marked), traverse the tree from Nt to the root until a node of which the corresponding node on the association graph has an unmarked arc is found. If there exists such a node, name the node as Nt and the corresponding node on the association graph as N. Go to 2. Otherwise, go to 3.
3. [Carrying ?variables to the root]
   For each ?variables, put it on all arcs on the pass from the node which contains the variable to the root. Terminate.

Remark. If the node M is marked in step 2, then the corresponding tree node Mt must be on the pass which connects the root with Nt. This fact is easily proved as follows. Since the tree is constructed in pre-order, all nodes are completely expanded except those (JV which the corresponding nodes, in the tree are on the pass. But since we reached the node M via an unmarked arc, it is expandable. Therefore, the node M is on the pass from Nt to the root.