EVALUATION OF DATABASE RECURSIVE LOGIC PROGRAMS AS RECURRENT FUNCTION SERIES

Georges GARDARIN
Christophe de MAINDREVILLE

SABRE Project
INRIA & University Paris VI
B.P. 105, 78153 Le Chesnay-Cédex (France)

ABSTRACT

The authors introduce a new method to compile queries referencing recursively defined predicates. This method is based on an interpretation of the query and the relations as functions which map one column of a relation to another column. It is shown that a large class of queries with associated recursive rules, including mutually recursive rules, can be computed as the limit of a series of functions. Typical cases of series of functions are given and solved. The solutions lend themselves towards either extended relational algebra or SQL optimized programs to compute the recursive query answers. Examples of applications are given.

1. INTRODUCTION

Assuming the reader to be familiar with deductive databases [GALL 84], we address the problem of evaluating queries referencing recursively defined relations in a deductive database context. Several solutions have been proposed, among them [CHAN 81, MARQ 83, HENS 84, ROHM 85, LOZI 85, ULLM 85, BANC 85c].

Solutions have been classified in two types [GALL 81] :

(i) Interpretation. This strategy is mainly derived from backward chaining and works in a top-down manner. It suffers from two principal drawbacks. First, the halting and completness conditions of the deductive process are not easy to determine. They lead the approach to be rather difficult to implement [VIEI 85]. Second, interpreted methods perform the inference basically one tuple at a time and do not take advantage of the efficient set manipulation primitives offered by relational DBMSs. Also, as these methods do not pre-compile the queries, they generate call loops to the DBMS which are rather inefficient. However, interpreted methods are able to take advantage of the constants specified in the query when these constants propagate to the base predicates . Thus, they efficiently perform selections at first in favorable cases, although they inefficiently work one tuple at a time in straightforward implementations.

(ii) Compilation. This approach is mainly derived from forward chaining and works in a bottom-up manner. The basic idea is to compile the query and rules in a relational algebra program (including loops) which can be executed by the DBMS. A general and efficient compilation algorithm must generate programs which present two interesting features [BANC 85a] :

(a) No redundancy, that is tuples must not be produced twice using the same rules. The method described in [BANC 85b] is an example of such an improvement.

(b) No generation of useless tuples, that is the constants which are given in the query must be used as soon as possible to eliminate tuples which are useless at the next steps of the derivation process. The method described in [CHAN 81] is an example of an efficient compilation method, but it only applies to regular rules. A uniform formalism called rule/goal graphs has been proposed to describe such methods [ULLM 85]. Recently, the elegant notion of magic set has been introduced to avoid generation of useless facts [BANC 85c].

In this paper, we first propose a formalism which consists in interpreting relations and queries as functions. The intuitive idea is that a relation instance defines functions, each function being the mapping from one set of values in one column to the

corresponding set of values in another column Then, we use this functional formalism to translate queries and rules into recurrent series of functions The resolution of the series appears then as a classical problem of mathematics The method is general and applies for a large set of rules including non linear and mutual recursive rules When the resolution of a serie is feasible, the solution may be translated into either an optimized SQL program or in a program of relational algebra operators extended with fixpoint operators, such as transitive closure or extended transitive closure We describe in details a SQL program generation method for linear rules Moreover, we introduce a new relational operator called external closure which permits to solve any acyclic linear rule, a kind of rules which are precisely defined below

The paper is organized as follows First, we introduce some background on recursive Horn clause evaluation This background is usefull in addition to [GALL 84] and limited notions of graph theory for a good understanding of our method Reading this background section may be omitted for the understanding of the method It is only usefull for the proof of certain theorems Then, we introduce the functional view of relations and queries which is the basis of the method In section four, we present the algorithm to translate recursive rules and the associated queries into recurrent series of functions Next, we present typical cases of functional equations corresponding to linear rules which are easy to solve and to translate into SQL programs or in extended relational algebra programs using the external closure operator which is defined Finally, we introduce possible extensions of the method

2. BACKGROUND ON RECURSIVE HORN CLAUSE THEORY

2.1 Logic program .

A logic program consists of a set of Horn clauses A set of Horn clauses can be used to derive a virtual relation from a relational database Such programs are composed of three sets of rules

(i) A first set of rules allows the system to derive non recursive relations S0, S1 Sp from the base relations B0, B1 Bn , certain of the Si may be base relations Bi , in this case the rule i is absent from this first set whose general form is as follows

S_1 <--- f1 (B0, B1, Bn)

S_2 <--- f2 (B0, B1, Bn)

S_p <--- fp (B0, B1, Bn)

(ii) A second set of rules gives initial values to the recursive predicates R_1, R_2 R_n as follows

R_1 <--- g^1_1 (S1,S2, , Sp)

R_1 <--- g^1_{s1}(S1,S2, , Sp)

R_2 <--- g^2_1(S1,S2, Sp)

R_2 <--- g^2_{s2}(S1,S2, Sp)

R_n <--- g^n_1(S1,S2, Sp)

R_n <--- g^n_{sn}(S1,S2, Sp)

(iii) Finally, the third set allows the system to compute recursively each R_i , with possibly mutual recursions

R_1 <--- h1 (R_1R_2, ,R_n, S1, S2, Sp)

R_2 <--- h2 (R_1R_2, ,R_n, S1, S2, Sp)

R_n <--- hn (R_1R_2, ,R_n, S1,S2, Sp)

In all the above rules, fi, gi and hi are relational algebra expressions of the given argument relations (some may be absent) For simplicity, we assume that all arguments of our predicates are variables , thus, fi, gi and hi are relational algebra expressions composed with joins and projections Every logic program can be cast into the given form by representing the constants as facts

It is known [CHAN 82] that the recursive relations which are derived from the given set of Horn clauses are the least fixpoints of the following system of equations

$R_1 = G1(S1,S2, ,Sp) \cup H1(R_1, R_n,S1, ,Sp)$

$R_n = Gn(S1,S2, ,Sp) \cup Hn(R_1, R_n,S1, ,Sp)$

where Gi, Hi are composed with joins, projections and unions of relations chosen among R_1, R_n, S1, S2, Sp

2 2 Least fixpoint operators

Let us consider a relation R defined recursively by an equation of the form

R = F (R) (1)

where F (R) is a relational expression with operand R, perhaps among other operands S1, S2 Sp F is a function over the relations having the same schema as R Such relations form a complete lattice using the set inclusion Thus, if F is increasing (i e R1 >= R2 implies F (R1) >= F (R2)), the Tarski theorem

[TARS 55] applies and we know that equation (1) has a set of solutions which is a complete lattice. Moreover, as all argument relations are finite, it has been demonstrated [AHO 79] (the demonstration is simply done by induction on i using the finitude hypothesis) that (∅ denotes the relation with no tuple):

(i) The sequence $F(\emptyset)$, $F(F(\emptyset))$, ... $F^n(\emptyset)$ is convergent towards a term denoted $\lim F^n(\emptyset)$.

(ii) $\lim F^n(\emptyset)$ is the least fixpoint of equation (1).

It is important to point out that every relational expression composed with union, join and projection is increasing [AHO 79]. Thus, the previous result directly applies to a recursive relation defined by a logic program.

This result leads to a very simple way to compute a recursive relation R as the least fixpoint of equation (1). This can be done using the naïve program [BANC 85a] :

R := F (∅) ;
while R changes do R := F (R);

An elegant optimization of this procedure using a differential approach can be found in [BANC 85b]. Nevertheless, such a procedure may be inefficient to solve queries because useless tuples may be generated. The difficult problem is to perform the possible selections derived from the query before applying the fixpoint computation algorithm. In the sequel, we propose a general solution to this problem.

3. RELATIONS AND QUERIES AS FUNCTIONS

3.1. Functions defined by a relation :

Let R (A1, A2, ... An) be a relation of attributes A1, A2 ... An. Let i and j be two subsets of {1, 2, ... n}. We shall use the notation Ai (resp. Aj) to denote the set of attributes indexed by the elements of i (resp. j). For example, with i = {2, 4}, Ai will designate A2, A4. In most cases, i and j will be subsets of one indice ; therefore, i (resp. j) may be seen as the rank of attribute Ai (resp. Aj) in the relation R. We denote dom (i) (resp dom(j)) the domain of Ai (resp. Aj), that is in general the cartesian product of the domains of the composing attributes. A given instance of R determines a function R :i→ j defined as follows .

Definition 1 : Relational function

A relational function R:i→ j is a function such that :
(i) The definition domain is the power set of dom (Ai), and the image domain is a subset of the power set of dom(Aj).
(ii) Let x = {x1, x2, ... xp} be a subset of dom(Ai);

R:i→j (x) is the subset y = {y1, y2, ... yq} of dom(Aj) obtained by restricting R to those tuples having x1 or x2 or ... xn for value of Ai, keeping only the values of Aj as a set :

$$y = \{\Pi_{Aj} (\sigma_{Ai = x1 \text{ or } \ldots \text{ or } Ai = xn} (R))\}.$$

Intuitively, R:i → j is the function which leads from a set of values of attributes Ai to the corresponding set of values of attributes Aj following the relation R.

Let us give examples of relational functions derived from the relation instance PARENT portrayed Figure 1.

PARENT	PARENT	CHILDREN	BIRTH-DATE
	lulu	toto	1970
	tintin	lulu	1945
	lili	toto	1970
	titine	lulu	1945

Figure 1. : An instance of the PARENT relation

For example the previous relation defines the following functions:

(i) PARENT:1 →2(x) which determines the children of given parents; for instance, we get:

PARENT:1→2({∅}) ={∅};

PARENT:1→2({lulu}) = {toto};

PARENT:1→2({lulu,tintin}) = {toto,lulu};

PARENT:1→2({lulu,tintin,totoche}) = {toto,lulu};

PARENT:1→2({totoche}) = {∅};

(ii) PARENT:2 →1(x) wich determines the parents of given children; for example, we get:

PARENT:2 →1({∅}) ={∅};

PARENT:2 →1({toto}) = {lulu,lili};

PARENT:2 →1({toto,lulu}) = {lulu,tintin,lili,titine};

PARENT:2 →1({toto,totoche}) = {lulu,lili};

(iii) PARENT:1→2,3(x) wich determines the children with birthdate for given parents; for example, we obtain:

PARENT:1→2,3({lulu}) = {toto-1970};

PARENT:1→2,3({lulu,tintin}) = {toto-1970,lulu-1945};

3.2. Sum and composition of relational functions:

We shall use the following classical operations over functions and function results:

(i) The union of two sets resulting from a function evaluation is denoted + ; We obviously have f(x+y) = f(x) + f(y).

(ii) The sum of two functions having same domain is defined by:

$$(f+g)(x) = f(x) + g(x).$$

(iii) The composition of f and g is possible if the image domain of g is included in the definition domain of f ; it is defined by:

$$(f \circ g)(x) = f(g(x)).$$

3.3 Selections as evaluation of functions

In this section, we consider possible queries on a relation using the comparators = , < , > , =< , >= , and the logical operators "or" and "and". We shortly show that such queries can be expressed simply as a function evaluation.

Lemma 1:

> Any selection request over a relation R can be represented as a sum of relational evaluations.

Proof:

A selection of a relation R can be expressed using relational algebra as $\Pi_{Aj}(\sigma_Q(R))$. The result being a set of Aj values, the image domain of the functions wich may be included in the sum is $2^{dom(Aj)}$. Any disjunctive restriction on attribute(s) Ai of a relation R(A1,A2,..An) followed by a projection on attribute(s) Aj can be expressed as a function's evaluation as follows:

$\Pi_{Aj}(\sigma_{Ai=c1 \text{ or...or } Ai=ck}(R)) = R: i \rightarrow j(\{c1,...,ck\})$

Any conjonctive restriction on attributes Ai1,Ai2,...,Aik (with Aip different from Aiq for all p,q, otherwise the result is empty and the empty fonction may be used) followed by a projection on attribute(s) Aj, can be expressed as a function's evaluation as follows :

$\Pi_{Aj}(\sigma_{Ai1=c1 \text{and...} Aik=ck}(R)) = R:i1,...,ik \rightarrow j(\{c1,...,ck\})$

More generally,it is always possible to put the qualification Q in disjunctive normal form. Each conjunction may then be replaced by a relational function's evaluation. When the θ comparator is not the equality, the cj corresponding set is defined by $\{Aj\ \theta\ cj\}$. The total query is then equivalent to the sum of all the relational function evaluations derived from each conjunction.♦

Let us give some examples of simple selections expressed as relational function's evaluation using the parent relation :

(i) give toto's parents :

PARENT : 2→1({toto}).

(ii) give toto's parent or lulu's parents :

PARENT : 2→1({toto,lulu}).

(iii) when does lulu become toto's parent ?

PARENT :1, 2→3({lulu-toto}).

(iv) give the parents who had children since 1983:

PARENT : 3→1({BIRTH-DATE >=1983}).

3.4 Semi-join as composition of functions

It is important to point out that the composition of two functions f and g derived from two relations R and S may be defined using a relation which is the join of R and S. More precisely, we can demonstrate the following lemma which shows that a function composition is really equivalent to a semi-join [BERN79].

Lemma 2 :

> Any sequences of the following relational algebra operations :
> (i) selection of a relation R on attribute(s) Ai,
> (ii) natural join of the result with relation S,
> (iii) projection of the result on attribute(s) Bq of S,
> can be represented as a composition of two relational functions derived from R and S.

Proof :

Let R(A1,...,An) and S(B1,...,Bm) be two relations.

Let : $E=\Pi_{Bq}(\sigma_{Ai=x}(R)$ 'join'$_{Aj=Bp}$ $S)$

Then, using the definition of the relational functions, we can demonstrate obviously :

$E = S : p \rightarrow q \circ R : i \rightarrow j\ (x)$. ♦

The following corollary of this lemma will be used in the sequel:

Lemma 3:

> The composition of two relational functions f o g can be expressed as a unique relational function using the join of two relations defining the relational functions f and g.

Proof :

Let $E = S : p \rightarrow q \circ R : i \rightarrow j\ (x)$ and let T = R 'join'$_{Aj=Bp}$ S

We can show by using the definition of the relational functions that

$$E = T_{1 \to q(x)} \quad \blacklozenge$$

4 EVALUATION OF RECURSIVE QUERIES

4.1 Recursive queries

A recursive query is a query on a recursively defined relation R Using section 3, it appears that the evaluation of a recursive query expressed in SQL as follows

SELECT A_j FROM R WHERE A_i = c

consists in evaluating the function

$$R_{i \to j(c)}$$

More generally, the evaluation of

SELECT Aj1, ,Ajk FROM R WHERE $A_i \theta$ c consists in the evaluation of the function

$R_{i \to j1, \ ,jk(\{A_i \theta c\})}$

Thus, recursive query evaluation leads to relational function's evaluations For simplicity, we shall restrict ourself to the case where A_j and A_i are unique attributes

4 2 Recursive relations as functions' series

Let R be a recursively defined relation As stated R is the least fixpoint of an equation R = F(R), where F is a relational algebra expression Using the notations R0=∅,R1=F(∅), , $Rn = F^n(\emptyset)$,the solution of the equation is R = lim Rn with Rn= F(Rn-1) In the sequel, we give sufficient conditions for any function $Rn_{i \to j}$ to be expressed as an expression of the function $Rn\text{-}1_{i \to j}$, possibly combined with other known functions, where the expression is composed with the addition (+) and the composition (o)

4.3 Connection graph of a rule

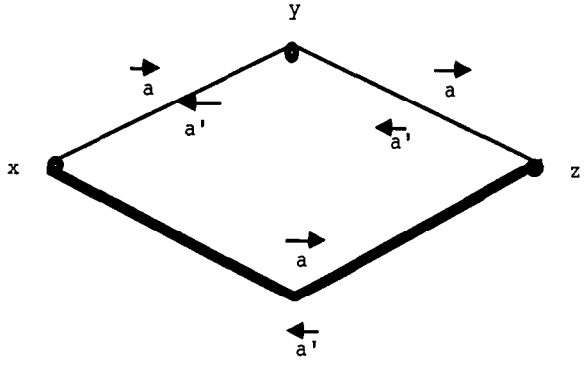

Figure 2 . example of a connection graph

Let us now introduce the notion of acyclic rule We call condition connection graph the connection graph corresponding to the condition of a rule, that is the rule connection graph without the distinguished arcs

Definition 3

An acyclic rule is a rule with an acyclic condition connection graph

Examples

The classical rules defining the recursive predicates Ancestors, Same-generation-cousins, Family-friends are acyclic The following rule is cyclic

Q(x,y) <--- B1(x,x1)&Q(x1,y)&B2(x,x2)&Q(x2,y)&B3(y,z)

To translate a recursive rule into recurrent functional equations, we use a connection graph It allows us to derive the set of all functional equations which are equivalent to a rule Such a graph is defined as follows

Definition 2. rule connection graph

A connection graph of a rule is a labelled graph such as

(i) A node corresponds to each variable appearing in the rule

(ii) For any pair of variables x and y appearing in the same predicate Q, there exists an arc x <--->y labelled by the corresponding relational functions $Q_{i \to j}$ and $Q_{j \to i}$

(iii) The labels are oriented according to the traversing direction

(iv) The arcs corresponding to the left occurrence of the predicat are special ones, portrayed by darker edges

To simplify labels, if j > i we shall note $Q_{i \to j}$ by q and $Q_{j \to i}$ by q' As we generally use binary predicates, this shall not lead to confusion

Figure 2 gives an example of a connection graph for the following rule A(x,z) <--- A(x,y) & A(y,z)

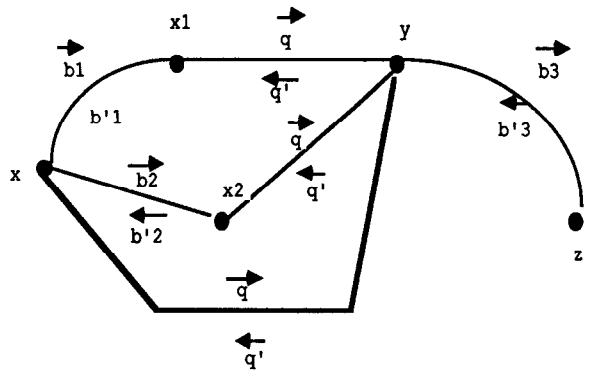

Figure 3 : cyclic connection graph

4 4 Translation of a recursive rule into a system of recurrent functional equations

In this section, we present a theorem wich gives sufficent conditions to deduce from the connection graph a system of functional equations We restrict ourself to the case of binary predicates with two variables Therefore, the connection graph has a unique distinguished edge with two extremities It is always possible to translate non binary predicates into binary ones

Theorem :

> A set of acyclic rules is equivalent to a system of functional equations if, for each rule, there exists an Euler path in the condition connection graph between the two extremities of the distinguished arc

Proof

The proof consists in giving an algorithm to derive the functional equations from the connection graph For each function $p_n(x)$ corresponding to a recursive predicate P, an equation $p_n(x) = E(x)$ is put down The contribution of a rule involving P in E is obtained by following the Euler path (that is a path which does not cross twice the same node) in the direction given by the evaluated function More precisely, the algorithm generating the equations is given below The equivalence means that evaluating the recurrent function limits for their definition domains gives the recursive predicate's instances This is due to the background recalled above and to the definition of the functional form of a relation ♦

Algorithm to built the system of equations.

For each rule associated to a predicate P(x,y)

1) Derive the equation $P_n(x) \supset E(x)$ by covering the Euler path from x to y where E is the composition of the functions labelling the Euler path from x to y, the recursive predicates being indiced by n-1

2) Derive the equation $P'_n(x) \supset E'(x)$ by covering the Euler path from y to x, E' being the composition of the functions labelling the Euler path from y to x

3) For each functions P_n (resp P'_n), collect the different equations and set $P_n = \Sigma E(x)$ (resp $P'_n = \Sigma E'(x)$)

4) Add the initialization equations $P_0 = \emptyset$ (resp $P'_0 = \emptyset$) ♦

Lemma 4:

> A query answer is given by resolving the system of equations generated by the above algorithm , i e the query P(x?,{a}) (resp P({a},y?) is answered by computing p({a}) = lim p_n({a}) (resp p'({a}) = lim p'_n({a})) with the system of equations

Proof

The limit may be computed recursively as

$p_0(\{a\}) = \{\emptyset\}$

$p_1(\{a\}) = \{E(p_0)\{a\}\}$

$p_n(\{a\}) = \{E(p_{n-1})\{a\}\}$

The existence of the limit is given by the Tarski's theorem We point out that the application of the function to the set {a} corresponds to focus on the restriction which can be any restriction predicate ♦

The evaluation method proposed in the above proof which performs restriction as soon as possible is very general in the sense that

1) The set {a} can be defined extensionally or intentionally For instance
   {a} = {Toto}, {a} = {Toto, Paul, Tintin}
   {a} = {y / 10 < y = < 25}
   are acceptable sets
2) We can generally express $p_n(x)$ using only $p_k(x)$ (with k< n) and thus we eliminate mutual recursions

4 5. Examples of applications:

(i) Ancestors

Let us consider the very classical rules where P represents the base relation PARENT and A the derived relation ANCESTOR

A(x,y) <--- P(x,y)

A(x,y) <--- A(x,z) & P(z,y)

The query SELECT PARENT FROM ANCESTOR WHERE CHILDREN = "toto" corresponds to A(toto,y?), i e to evaluate the function $a = A\ 1 \rightarrow 2(x)$, with x = {toto}

We note $P\ 1 \rightarrow 2(x) = p(x)$, $P\ 2 \rightarrow 1(x) = p'(x)$

$A\ 2 \rightarrow 1(x) = a'(x)$

Using the connection graph and the above algorithm, we get the system

$a_0(x) = \varnothing \qquad a_n(x) = p \circ a_{n-1}(x) + p(x)$

$a'_0(x) = \varnothing \qquad a'_n(x) = a'_{n-1}(x) \circ p'(x) + p'(x)$

Keeping the pertinent equations for solving the query, it yields

$a_0(x) = \varnothing$

$a_n(x) = p \circ a_{n-1}(x) + p(x)$

that we must evaluate for x = {toto}

(ii) Family-friends

Let us consider the base relations PARENT (P) and FRIEND (F) and a derived relation FAMILY-FRIEND (FF) wich represents the friends of the ancestors defined as follows

FF(x,y) <--- F(x,y)

FF(x,z) <--- P(x,y) & FF (y,z)

The query FF (toto,y?) corresponds to evaluate the function

$FF\ 1 \rightarrow 2(x) = ff(x)$ with x = {toto}

We note $f(x) = F\ 1 \rightarrow 2(x)$ and $p(x) = P\ 1 \rightarrow 2(x)$

The pertinent equations are

$ff_0(x) = \varnothing$

$ff_n(x) = f(x) + ff_{n-1}(p(x))$

to evaluate for x = {toto}

(iii) Cousins of the same generation (non-stable case)

Let us assume the base relation PARENT (CHILDREN , PARENT) and a derived relation HUMAN (NAME) which can be derived from the PARENT relation by the following non recursive axioms

HUMAN(x) <--- PARENT(x,y)

HUMAN(y) <--- PARENT(x,y)

The purpose of the example is to compute the recursive predicate cousin of the same generation (SG) [BANC 85c] defined as follows from the HUMAN (H) and PARENT (P) predicates (the example is non stable because we flip x_p and y_p in the right occurence of SG)

SG(x,y) <--- H(x) (1)

SG(x,y) <--- P(x,xp) & SG(yp,xp) & P(y,yp) (2)

The considered query consists to find the same generation's cousins of "Agamemnon", that is SG(Agamemnon,?y)

This leads us to evaluate the function

$SG\ 1 \rightarrow 2(x)$ for x = {Agamemnon}

The connection graph is

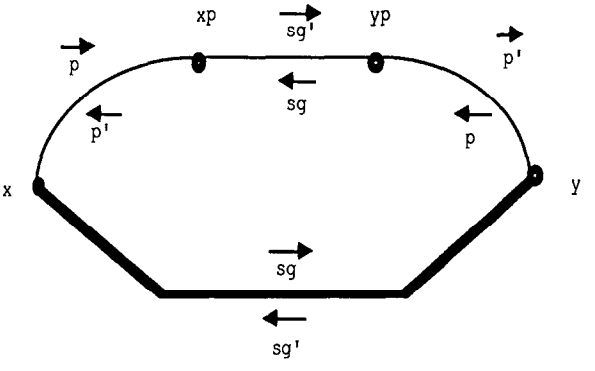

Figure 4 · Connection graph for the same generation cousins

The equations are (1_h being the identity for Human)

(1) $sg_0(x) = \varnothing \qquad sg'_0 = \varnothing$

(2) $sg_n(x) = p'(sg'_{n-1}(p(x))) + 1_h(x)$

(3) $sg'_n(x) = p'(sg_{n-1}(p(x))) + 1_h(x)$

We have to evaluate $sg_n(x)$ Eliminating sg'_{n-1} in equation (2) using equation (3), we get

$sg_0(x) = \varnothing$

$sg_n(x) = p'(p'(sg_{n-2}(p(p(x))))) + p'(p(x)) + 1_h(x)$

The answer to the query is obtained by the substitution x /Agamemnon

5 SOLVING LINEAR RULES

5 1 Linear systems .

In this section we address the problem of the resolution of the series of functions and more precisely the particular case of

linear systems Linear systems are derived from linear rules, that is rules in wich the left predicate is used only once in the right part of the rule

Definition 4 Linear systems

A linear system is a set of rules which leads for a recursive query to a functions' series of the form

$r_0(x) = \varnothing$

$r_n(x) = s(x) + q(r_{n-1}(t(x)))$

where s, q, t are functions derived from non recursive predicates

We call regular systems a subclass of the linear one where t is the identity function They correspond to rules processed in [CHAN81] Linear systems are more general The examples presented above are all linear All the sets of acyclic rules where the recursive predicate appears only once in the condition part of the rule leads to linear systems Let us point out that systems where the second equation is of the form

$$r_n(x) = s(x) + q(r_{n-k}(t(x)))$$

where k is any positive integer are equivalent to linear systems This is because we are only looking at lim $r_n(x)$ Thus, we include such systems in the linear ones

The linear systems can be simply solved as follows, using the notation q^n to represent q o q o q (n times)

$r_1(x) = s(x)$

$r_2(x) = s(x) + q(s(t(x)))$

$r_3(x) = s(x) + q(s(t(x))) + q^2(s(t^2(x)))$

Let us assume the induction hypothesis

$$r_{n-1}(x) = s(x) + q(s(t(x))) + q^2(s(t^2(x))) + \ldots + q^{n-2}(s(t^{n-2}(x)))$$

With the functions additivity and the induction hypothesis, we can assert for all n

$$r_n(x) = s(x) + q(s(t(x))) + q^2(s(t^2(x))) + \ldots + q^{n-1}(s(t^{n-1}(x)))$$

5.2. Resolution by iterative programs

The limit of the polynom defined above can be computed by the following program

Procedure EVALUATE (R,x),

```
i,n integer,
Begin
   n = 0,
   R = s(x),
   PROC = ∅,
   DT = t(x),
   While DT ≠ ∅ do
         n = n+1,
         DR = s(DT),
         for i = 1 to n do
           DR = q(DR), od
         R = R + DR,
         PROC = PROC + DT,
         NDT = t(DT),
         DT = NDT - PROC,
   od,
end
```

The virtual relation R contains the result after the procedure's execution, x is the restriction criteria, Let us point out once more that x can be any set of values defined in intention or extension This procedure applies to cyclic relations because the PROC set variable memorizes the processed tuples The above presented examples correspond to acyclic databases, that is T is an acyclic relation In this case, we don't have to memorize the processed tuples as we never get them again (ie PROC is useless) The previous result can be applied to the family's friends example The series obtained was a linear one as defined above where rn is called ffn , s is denoted f , q is the identy and t is denoted p wich is an acyclic function Therefore, the procedure to compute ff(x) is

Procedure evaluate (FF,x) ,

```
i,n integer,
  Begin
     n = 0,
     FF = f(x),
     DP = p(x),
     while DP ≠ ∅ do
        n = n+1,
        DFF = F(DP),
        FOR i = 1 TO n do DFF = DFF od,
        FF = FF + DFF,
        DP = p(DP),
     od,
  end,
```

As q is the identity function the loop to build q (DFF) is obviously useless Therefore, we get the simplified procedure to retrieve the family's friends of x

Procedure evaluate (FF,x) ,

Begin
FF = f(x) ,
DP = p(x) ,
while DP ≠ ∅ do
DFF = f(DP) ,
FF = FF + DFF ,
DP = p(DP) ,
od ,
end ,

We can move back to relational algebra using SQL notations Thus, we get the following procedure to evaluate the family's friends of x (where x may be toto or any set of people)

Procedure evaluate (FF,x) ;

Begin
FF = SELECT PERSON2 FROM FRIEND
WHERE PERSON1 = x ,
DP = SELECT PARENT FROM PARENT
WHERE CHILDREN = x ,
while DP ≠ ∅ do
DFF = SELECT PERSON2 FROM FRIEND
WHERE PERSON1 IN DP ,
FF = FF UNION DFF ,
DP = SELECT PARENT FROM PARENT
WHERE CHILDREN IN DP ,
od ,
end ,

5 3 Resolution by a specific operator

A most optimal solution to compute linear systems is to implement a specific operator that we call external closure operator

Definition 5 External closure

The external closure operator is given by the formula

$$\Phi(Q,S,T,x) = \bigcup_{i} Q^i S T^i(x)$$

External closure is an operator which closes a relation T (after a possible restriction) at level n, then joins the result with a relation S, then performs n joins with a relation Q, finally projects on S schema and collects the results for every n including the initial relation S Such an operator is exactly what is needed to compute the answers of linear systems We could also distinguish left external closure where a relation S is composed at left by Q, and right external closure where it is composed at right by T

6 EXTENDING THE METHOD

Our approach can be extended in several ways

(i) First, the Euler condition on the connection graph is too strong It is sufficient to the path between the root and the target being a tree , that is always the case with acyclic rules The pending edges of the tree may be reduced by the composition of the labelling functions

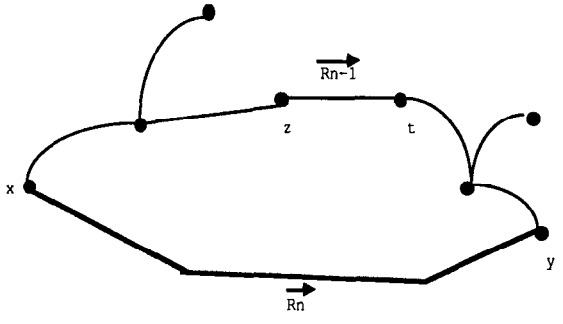

Figure 6 . Example of an acceptable connection graph

(ii) It is possible to deal with cyclic rules by using a relaxation/certification method It consists ,in a first relaxation step, to eliminate some arcs of the connection graph in order to make it acyclic The generated system is then computed and gives more solutions than needed, because of the suppression of a constraint The exact solution is obtained by a second certification step which consists in executing the rules, starting with the precedent solution set in a forward way

To conclude this section and show the generality of the method, let us give an example with two mutually recursive predicates

Let us consider the following set of rules

R1(x,y) <--- B4(x,y) R2(x,z) <--- B1(x,y) & R1(y,z)
R2(x,y) <--- B2(x,y) R1(z,x) <--- R2(x,y) & B3(z,y)

The system of equations equivalent to these rules is

$r1_n(x) = b4(x) + r2'_{n-1}(b3(x))$

$r1'_n(x) = b4'(x) + b3'(r2_{n-1}(x))$

$r2_n(x) = b2(x) + r1_{n-1}((b1(x))$

$r2'_n(x) = b2'(x) + b1(r1'_{n-1}(x))$

Let the query R2(a,y?) we have to evaluate lim $r2_n(a)$

It is obviously possible to express $r2_n$ in function of $r2_{n-4}$ and base relations as follows

$$r2_n(a) = b2(a) + b4(b1(a)) + b2'(b3(b1(a))) + b1(b4'(b3(b1(a)))) + b1(b3'(r2_{n-4}(b3(b1(a)))))$$

Indeed, this is a linear series that one may solve as shown above

7 CONCLUSION

In this paper, we have presented a framework for compiling a large class of recursive queries into SQL optimized programs or extended relational operations The concerned class includes all the linear, non linear, mutually recursive, non-stable rules as defined by other authors

We claim that the approach is easy to implement for at least linear systems The approach allows us to isolate a relational operator called external closure which computes all the linear acyclic rules

There is much more work that can be done to generalize such operators with complex conditions on join operations (with < or > comparators), arithmetic computations at each step or with functions included as predicate parameters We believe that the efficient implementation of such operators is necessary to process efficiently general and applicable recursive queries

Acknowledgments

We gratefully acknowledge the SABRE team for helpfull discussions and more specially Eric SIMON for his incisive comments on an earlier version of this paper We would like also to thank François BANCILHON for sending us some very nice papers

REFERENCES

[AHO 79] AHO A V , ULLMAN J D "Universality of data retrieval languages",Conf of POPL,San-Antonio, Texas, 1979

[BANC 85a] BANCILHON F "Evaluation des clauses de Horn dans les bases de données relationnelles", A paraître dans PRC BD3, Eyrolles 1986

[BANC 85b] BANCILHON F "Naive evaluation of recursively defined predicates" MCC internal report, 1985

[BANC 85c] BANCILHON F , MAIER D , SAGIV Y , ULLMAN J D "Magic sets and other strange ways to implement logic programs", MCC internal report, 1985

[BERN 79] BERNSTEIN P A , CHIU D W "Using semi-joins to solve relational queries", Technical report CCA 01-79, Jan 1979

[CHAN 82] CHANDRA K A , HAREL D "Horn clauses and the fixpoint query hierarchy", Proc ACM Symp on principles of databases, 1982

[CHAN 81] CHANG C "On evaluation of queries containing derived relation in a relational database", in [GALL 81]

[GALL 81] GALLAIRE H , MINKER J , NICOLAS J M "Advances in database theory", Book, Vol 1, Plenum Press, 1981

[GALL 84] GALLAIRE H , MINKER J , NICOLAS J M "Logic and databases a deductive approach", ACM Computing Surveys, Vol 16, N° 2, June 1984

[HENS 84] HENSCHEN L J , NAQVI S A "On compiling queries in recursive first-order databases", JACM, Vol 31, N° 1, Jan 1984

[LOZI 85] LOZINSKII E L "Evaluation queries in deductive databases by generating subqueries", IJCAI Proc , pp 173-177, Los Angeles, August 1985

[MARQ 83] MARQUE-PUCHEU G "Algebraic structure of answers in a recursive logic database", Rapport Ecole Normale Supérieure, 1983

[ROHM 85] ROHMER J , LESCOEUR R "La méthode d'Alexandre une solution pour traiter les axiomes récursifs dans les bases de données déductives", Rapport de recherche, Bull, DRAL/IA/45 01 1985

[TARS 55] TARSKI A "A lattice theoretical fixpoint theorem and its applications", Pacific journal of mathematics, N° 5, pp 285-309, 1955

[ULLM 85] ULLMAN J D "Implementation of logical query languages for database", ACM SIGMOD, Austin, Texas, 1985

[VIEI 85] VIEILLE L "Recursive axioms in deductive databases the query sub-query approach", ECRC internal report 1985