INFORMATIQUE THÉORIQUE ET APPLICATIONS

IRÈNE GUESSARIAN

A note on fixpoint techniques in data base recursive logic programs

Informatique théorique et applications, tome 22, n° 1 (1988), p. 49-56

<http://www.numdam.org/item?id=ITA_1988__22_1_49_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 22, n° 1, 1988, p. 49 à 56)

A NOTE ON FIXPOINT TECHNIQUES IN DATA BASE RECURSIVE LOGIC PROGRAMS (*)

by Irène GUESSARIAN (1)

Communicated by J. E. PIN

Abstract. – *We show how to use fixpoint techniques to answer some non-linear recursive queries, by transforming such queries into left-linear (or regular) queries having the same solutions, and which can be answered via a while loop.*

Résumé. – *Nous montrons comment le calcul d'un plus petit point fixe judicieusement choisi permet de résoudre une requête récursive non linéaire : la méthode est de construire une requête linéaire à gauche (ou régulière) qui admette les mêmes réponses, lesquelles sont alors calculables par une boucle tant que.*

1. INTRODUCTION

Recursive predicates on data bases can be represented as recursive logic programs, i. e. as recursive systems of equations on relations. It is shown in [Gardarin, de Maindreville] that queries referring to such predicates can be represented as recurrent function series; then, [Gardarin, de Maindreville] uses fixpoint techniques to solve the recursive equation defining the series corresponding to the queries.

We build upon this approach and go further in the use of fixpoint techniques: namely, we directly use the fixpoint techniques (for *relations* instead of functions) to solve the recursively defined predicate on the data base; then we deduce the answers to queries from the solution of the relation defining the corresponding predicate. We show that combining the power of fixpoint techniques with the properties of relations can lead to very simple syntactic solutions to such recursive queries, wherefrom we can deduce immediately

(*) Received October 1986, revised March 1987.
This work was done while the author was visiting the University of Salerno.
(1) L.I.T.P., Université Paris-VII, 2, place Jussieu, 75251 Paris Cedex 05.

Informatique théorique et Applications/Theoretical Informatics and Applications
0296-1598 88/01 49 8/$2.80/

easy algorithms to compute an actual semantic query, as soon as the data corresponding to the query are given. One such example treated here, which generalizes one of the results in [Gardarin, de Maindreville] shows a quite general case in which a polynomial system can be transformed into a regular one; we thus obtain a simple iteration instead of a polynomial recursion, which has the advantage that the iteration can be solved by a classical while loop very easily. We think that such methods could be more widely used to solve efficiently recursively defined queries.

This paper is organized as follows: we first show how recursively defined queries lead to fixpoint equations, for relations or for functions (*cf.* [Gardarin, de Maindreville]); we then deal with the case of polynomial queries which are simple polynomial, and show how to use fixpoint techniques to solve them. The present paper is self-contained; however, for motivations and background on data base theory, the reader may consult [Aho-Ullman, Bancilhon-Ramakrishnan, Chandra-Harel, Gallaire-Minker-Nicolas].

2. QUERIES ON RECURSIVELY DEFINED RELATIONS AND FIXPOINT TECHNIQUES

In relational data bases, queries can be defined recursively, and viewed as relations on the columns of the data base. The following example illustrates this fact.

Example 1 : Consider the PARENT data base, which is described by a two column relation, the columns "parent" and "child"; an instance of this relation is depicted below:

PARENT:	*parent*	*child*
	John	Mary
	Mary	Peter Jr.
	David	Peter
	Peter	Peter Jr.

Let *PARENT* (*parent*, *child*) be the base relation denoted by P. Define now the *ancestor* relation on *parent* × *child* by:

$$A(x, y) \Leftarrow P(x, y)$$
$$A(x, y) \Leftarrow P(x, z) \wedge A(z, y)$$

In PARENT, let us assume the query: *select ancestor where child* = *Peter Jr.*, or, in logical terms, *ancestor* (*?*, *Peter Jr.*). For a relation R, let us define the

following abbreviation:

$$r(X) = R(?, X) = \{ y \mid R(y, x) \text{ for some } x \in X \}$$

Letting then $p(X) = P(?, X)$ and $a(X) = A(?, X)$, the above defined query is described as $a(Peter\ Jr.)$; now, a is defined in terms of base relations by the following equation:

$$a(x) = p(x) + p(a(x))$$

This is a recursive equation defining the relation a, which then can be solved by standard fixpoint techniques. Such techniques extend straightforwardly from solving equations defining functions to solving equations defining relations, as we will see.

DEFINITION 1: The relation recursively defined by a function F on a data base is the least relation $R \subset D = D_1 \times \ldots \times D_n$ which satisfies the equation $R = F(R)$, where F is a function mapping a relation R on D into another relation on D, i. e. $F : \mathcal{P}(D) \to \mathcal{P}(D)$.

Now, D ordered by inclusion, is a complete lattice; henceforth [Birkhoff], any monotone F, namely any F such that: $R \subset R' \Rightarrow F(R) \subset F(R')$, has a least fixpoint R_F; if moreover, either F is continuous, or the domain of F is finite (which is always true for data bases), then, R_F can be inductively computed as $\sup \{ F^n(\varnothing) \mid n \in N \}$. Now, we have the following:

PROPOSITION 1 [Aho, Ullman]: *Any function F (on relations) defined by using only* $\cup$ *also denoted* + *(union),* $\cap$ *(intersection), projections and compositions is monotone.*

Proof: By induction, the composition of monotone functions being monotone.

PROPOSITION 2 [Gardarin, de Maindreville]: *Any query defined by using only equality and the logical connectives* $\wedge$ *(and) and* $\vee$ *(or) can be expressed as a relation recursively defined by a monotone function.*

In fact, one can easily show that the queries considered in Proposition 2 and the functions considered in Proposition 1 are even continuous; henceforth, our results will apply also to the case where the domains would be infinite.

Most usual queries satisfy the hypotheses of Proposition 2, hence can be solved very easily using fixpoint techniques. In the next section, we show how to use such techniques to compute efficiently answers to recursively defined queries.

3. REGULAR QUERIES AND POLYNOMIAL QUERIES

In the sequel, and unless otherwise specified, relation (resp. query) will mean relation (resp. query) defined by a recursive equation. The simplest type of such relations or queries are the regular ones:

DEFINITION 2: (i) A query is said to be *regular* if the associated recursively defined equation is, namely iff the query is the solution of an equation of the form

$$r(x)=s(x)+q(r(x)) \tag{1}$$

(ii) A relation is said to be *regular* if the associated recursively defined equation is, namely iff the relation is the solution of an equation of the form

$$R=S+Q.R \tag{2}$$

where

S and Q are relations defined by non recursive expressions of base relations,

s, q, r are as previously the multivalued functions associated with S, Q, R,

$+$ denotes set-theoretic union,

$Q.R$ denotes the usual composition of relations, namely:

$$(x, y)\in Q.R \iff (x, z)\in Q \quad \text{and} \quad (z, y)\in R \quad \text{for some } z$$

Example 1 (continued): The ancestor relation and the query associated with it are regular.

PROPOSITION 3: *The solution of the regular equation* (2) *is*

$$R_{S,Q}=\sup\{R_n \mid n\in N\},$$

where $R_n=S+Q.S+\ldots+Q^n.S$ *and* Q^n *denotes* Q *composed with itself* n *times.*

Proof: The function F defined by $F(R)=S+Q.R$ is monotone by Proposition 1.

COROLLARY: *The solution of the regular equation* (1) *is*

$$r_{s,q}(x)=\sup\{r_n(x) \mid n\in N\},$$

where $r_n(x)=s(x)+q(s(x))+\ldots+q^n(s(x))$.

Since the solutions to queries can be derived from the description of the corresponding relation, we will choose to solve the equation defining the

relation first, and therefrom we will deduce the solutions of the various queries corresponding to that relation. Our approach slightly differs from the one of [Gardarin, de Maindreville] who directly solve the functional equation corresponding to the query. The main difference rests in the domains in which least fixpoints are computed: [Gardarin, de Maindreville] use as domain the lattice of elements of the data base, whereas we use as domain the lattice of relations over the data base. We will see how adopting our more general approach and allowing ourselves the flexibility of working also in the algebra of relations, can greatly simplify the solution of some queries. However, for the actual computation of the solution, we go back to the method of "propagation of selections" and compute only the values needed.

DEFINITION 3: A relation is said to be *polynomial* (*resp. simple polynomial*) if the associated recursively defined equation is, namely iff the relation is the solution of an equation of the form

$$\text{(polynomial) } R = S + Q_1 . R + Q_2 . R^2 + \ldots + Q_k . R^k \tag{3}$$

$$\text{(simple polynomial) } R = S + R^k \tag{4}$$

where S and Q_i for $i = 1, \ldots, k$ are as in Definition 2.

Example 2 [Gardarin, de Maindreville]: The relation AA, *ancestor of even generation*, is defined on *parent* × *child* of the data base PARENT, by:

$$AA(x, y) \Leftarrow P(x, z) \wedge P(z, y)$$

$$AA(x, y) \Leftarrow AA(x, z) \wedge AA(z, y)$$

hence, the relation AA satisfies the equation $AA = P^2 + AA^2$ and, letting the query: $aa(x) = AA(?, x)$, the solution of that query will be given by the equation: $aa(x) = p^2(x) + aa^2(x)$. The relation *ancestor of even generation* and the associated query are thus simple polynomial.

THEOREM 1: *Every simple polynomial relation* (*resp. query*) *is equivalent to a regular relation* (*resp. query*).

This theorem is proved by fixpoint techniques, namely by constructing for each simple polynomial query a regular query which has the same solution. It provides an extremely efficient way of answering simple polynomial queries. We need first some simple lemmas on relations.

LEMMA 1: *Let S be a relation; let $\mathbf{1}$ denote the identity relation, namely: $(x, y) \in \mathbf{1} \Leftrightarrow x = y$, $+$ denote the set-theoretic union and R^p denote the composition of relation R with itself p times; then:*

$$(\mathbf{1}+S)^n = \mathbf{1}+S+ \ldots +S^n$$

Proof: By induction on n. It is trivially true for $n=1$. Notice first that $+$ is idempotent, i. e. $S+S=S$. Assume $(\mathbf{1}+S)^n = \mathbf{1}+S+ \ldots +S^n$; then:

$$\begin{aligned}(\mathbf{1}+S)^{n+1} &= (\mathbf{1}+S)(\mathbf{1}+S)^n \\ &= \mathbf{1}+S+ \ldots +S^n+S(\mathbf{1}+S+ \ldots +S^n) \\ &= \mathbf{1}+S+ \ldots +S^n+S+ \ldots +S^{n+1} \\ \text{by commutativity of } + &: = \mathbf{1}+(S+S)+ \ldots +(S^n+S^n)+S^{n+1} \\ \text{by idempotence of } + &: = \mathbf{1}+S+ \ldots +S^{n+1}\end{aligned}$$

We can now solve equation (4) by fixpoint induction.

LEMMA 2: *The solution of equation* (4) *with $k \geqq 2$ is defined by: $R_\infty = \sup\{R_n \mid n \in N\}$, where $R_1 = S$ and for $n \geqq 2$, $R_n = S(\mathbf{1}+S^{k-1})^{p(n)}$ with $p(2)=1$, $p(n)=1+k+ \ldots +k^{n-2}$ for $n \geqq 3$.*

Proof: R_∞ is the least fixpoint of the function F defined by $F(R)=S+R^k$. It thus suffices to check by induction that $F^n(\varnothing)=R_n$. For $n=1$, 2:

$$F(\varnothing)=S$$
$$F^2(\varnothing)=S+S^k=S(\mathbf{1}+S^{k-1})$$

Let $n \geqq 2$ and assume $F^n(\varnothing)=R_n$, then:

$$\begin{aligned}F^{n+1}(\varnothing) &= S+R_n^k \\ &= S+S^k(\mathbf{1}+S^{k-1})^{kp(n)} \\ &= S(\mathbf{1}+S^{k-1}(\mathbf{1}+S^{k-1})^{kp(n)}) \\ &= S(\mathbf{1}+S^{k-1}(\mathbf{1}+S^{k-1}+ \ldots +S^{(k-1)kp(n)})) \text{ by Lemma 1} \\ &= S(\mathbf{1}+S^{k-1}+ \ldots +S^{(k-1)(kp(n)+1)}) \\ &= S(\mathbf{1}+S^{k-1}+ \ldots +S^{(k-1)p(n+1)}) \\ &= S(\mathbf{1}+S^{k-1})^{p(n+1)} \text{ by Lemma 1}\end{aligned}$$

LEMMA 3: *Let R_n be defined as in Lemma 2 and let*

$$T_n = S(\mathbf{1}+S^{k-1})^n = S(\mathbf{1}+S^{k-1}+ \ldots +S^{n(k-1)}).$$

Then:

$$R_\infty = \sup\{R_n \mid n \in N\} = \sup\{T_n \mid n \in N\} = T_\infty.$$

Proof: Clearly, since $k \geqq 2$, for n large enough, we will have $T_n \leqq R_n$, for all n, hence $T_\infty \leqq R_\infty$; conversely, for all n, $R_n \leqq T_{p(n)}$, hence $R_\infty \leqq T_\infty$.

Proof of theorem 1: If $k=0$ or $k=1$, the equation (4) is equivalent to the constant equation $R=S$, which is trivially regular. If $k \geqq 2$, then the sequence T_n is the solution of the regular equation $T=S+S^{k-1}.T$, which is thus equivalent to equation (4).

Theorem 1 provides a nice and easy solution to some classical non-linear queries such as the transitive closure T:

$$T(x, y) \Leftarrow P(x, y)$$

$$T(x, y) \Leftarrow T(x, z) \wedge T(z, y)$$

This result generalizes the result stated without proof in [Gardarin, de Maindreville] for the case $n=2$; Theorem 1 also solves one case of polynomial queries, thus partially answering a question of [Gardarin, de Maindreville]. More fundamentally:

1. We think that the method here employed, namely fixpoint techniques for *relations* instead of functions in conjunction with equivalences of fixpoints, can be fruitfully applied to solve other problems of recursive queries. Techniques borrowed from graph theory help in solving such problems [Gardarin, de Maindreville].

2. Since regular iterations are the simplest to solve, this provides a very useful tool for solving recursive queries.

3. The present approach complements the current trend towards solving queries at the functional level, with the actual data: we first solve at the *syntactic* and most general level for the relations, but staying at the level of symbolic computations, without doing the actual computations; then, once we have the data and want to do the actual computation, we come back at the semantic level, and deduce from our syntactic solution a simple algorithm, a while loop in the case of the present example, which computes the function solving the actual query. Hence, our method is in the mainstream of the methods "propagating the selections" [Bancilhon-Beeri *et al.*, Gardarin and de Maindreville].

ACKNOWLEDGMENTS

I thank the referee and D. Kfoury for helpful comments.

REFERENCES

1. A. V. AHO and J. D. ULLMAN, *Universality of Data Retrieval Languages*, Proc. Sixth A.C.M. Symp. on principles of programming languages, San Antonio, 1979, pp. 110-120.
2. F. BANCILHON, C. BEERI, P. KANELLAKIS and R. RAMAKRISHNAN, *Bounds on the Propagation of Selection into Logic Programs*, Proc. A.C.M. Symp. on principles of data base systems, San Diego, 1987.
3. F. BANCILHON and R. RAMAKRISHNAN, *An Amateur's Introduction to Recursive Query Processing Strategies*, Proc. A.C.M. Sigmod Conf., Washington, 1986.
4. G. BIRKHOFF, *Lattice Theory*, A.M.S., 1967.
5. A. CHANDRA and D. HAREL, *Structure and Complexity of Relational Queries*, Jour. Comput. Sys. Sci., Vol. 25, 1982, pp. 99-128.
6. H. GALLAIRE, J. MINKER and J. M. NICOLAS, *Logic and Data Bases: A Deductive Approach*, Assoc. Comput. Mach. Comput. Surveys, Vol. 16, 1984, pp. 153-185.
7. G. GARDARIN and C. DE MAINDREVILLE, *Evaluation of Data Base Recursive Logic Programs as Recurrent Function Series*, Proc. A.C.M. Sigmod Conf., Washington, 1986.