A GENERALIZATION OF THE DIFFERENTIAL APPROACH TO RECURSIVE QUERY EVALUATION

I. BALBIN AND K. RAMAMOHANARAO

▷ The differential (or seminaive) approach to query evaluation in function free, recursively defined, Horn clauses was recently proposed as an improvement to the naive bottom-up evaluation strategy. In this paper, we extend the approach to efficiently accommodate n recursively defined predicates in the body of a Horn clause. ◁

1. INTRODUCTION

The differential (or seminaive) approach to query evaluation is described in [2], [3], [5]. Focusing on the naive bottom-up method of evaluation, the approach eliminates some of the obviously redundant joins. However, "the problem is not solved in its entirety and only a number of transformations are known" [4]. In this paper, we extend the differential approach to include nonlinear rules with n recursively defined predicates in the body.

2. PRELIMINARIES

2.1. A Naive Execution Model

The idea of translating a recursive clause or view into an iterative equivalent is not new (see [6], for example). Termination is assured due to the finiteness of the database, and when the clauses are defined using only positive literals, their monotonicity ensures the computation of the least fixed point [1].

For the case of nonhierarchical databases, the problem of looping becomes an issue. The solution proposed in [7], with respect to their active connection graphs,

Address correspondence to I. Balbin or K. Ramamohanarao, Department of Computer Science, University of Melbourne, Parkville, Victoria 3052, Australia.

Received 23 June 1986; accepted 12 January 1987

THE JOURNAL OF LOGIC PROGRAMMING

52 Vanderbilt Ave., New York, NY 10017 0743-1066/87/$3.50

amounts to checking whether *all* tuples generated at each iteration of the loop are new. If this is not the case, the process is halted. For example, consider the iterative program corresponding to the following database which computes the transitive closure of p:

Example 1 (Transitive closure of p).

$$s(X,Y) \leftarrow p(X,Y)$$
$$s(X,Y) \leftarrow p(X,Z) \wedge s(Z,Y)$$

It can be viewed as computing

$$s_1(X,Y) = p(X,Y),$$
$$s_2(X,Y) = s_1(X,Y) + \Delta s_1(X,Y),$$
$$\vdots$$
$$s_{j+1}(X,Y) = s_j(X,Y) + \Delta s_j(X,Y) \quad \text{for} \quad j > 0,$$
$$\Delta s_j(X,Y) = p(X,Z) \bowtie s_j(Z,Y),$$

where $p(X,Y)$ is a shorthand for $\{(X,Y) | p(X,Y)$ is proved$\}$, and, for any predicate $s(X,Y)$, $s_j(X,Y)$ is defined as the set of s tuples which have been proved at the completion of the jth iteration; $\bowtie$ is the usual join operator, and $+$ is the set union. Δs_j is then the set of (possibly) new s tuples which have been formed by evaluating the join between p and s_j.

This type of program, whilst attractive in the sense that it does not go into an infinite loop, is fraught with some serious inefficiencies which are addressed by the differential approach.

3. THE DIFFERENTIAL APPROACH

3.1. Linear Horn Clauses

Using the method of [5],[3], at iteration j, rather than evaluating Δs_j by forming the join between $p(X,Z)$ and $s_j(Z,Y)$, we compute

$$\Delta s_j(X,Y) = p(X,Z) \bowtie \delta s_{j-1}(Z,Y)$$

where

$$\delta s_{j-1}(X,Y) = \Delta s_{j-1}(X,Y) - s_{j-1}(X,Y)$$

and, $-$ denotes set difference. Thus, δ depicts the *new* tuples generated at that iteration.

3.2. Extension to Nonlinear Horn Clauses

Ignoring variables, consider a recursive rule

$$s \leftarrow c \wedge a \wedge w$$

where

$$c \leftarrow c^1 \wedge c^2 \wedge \cdots \wedge c^m$$

and the c^k, $k = 1, \ldots, m$, are nonrecursively defined predicates; and a and w are recursively defined.

We write ca_jw_j to represent the set of all ground instances for which $c \wedge a \wedge w$ is true at the jth iteration, that is, $c \bowtie a_j \bowtie w_j$.

Now, for $j \geq 0$

$$s_{j+1} = s_j + \delta s_j, \tag{0}$$

where

$$\delta s_j = ca_jw_j - s_j.$$

Replacing a_j by its equivalent differential definition

$$a_j = a_{j-1} + \delta a_{j-1},$$

we obtain

$$\delta s_j = \left[c\left(a_{j-1} + \delta a_{j-1}\right)w_j\right] - s_j.$$

Since $\bowtie$ distributes over $+$, this simplifies to

$$\delta s_j = \left(c\,\delta a_{j-1}w_j + ca_{j-1}w_j\right) - s_j.$$

Similarly, by replacing the second occurrence of w_j with its equivalent differential definition

$$w_j = w_{j-1} + \delta w_{j-1},$$

we have

$$\delta s_j = \left[c\,\delta a_{j-1}\,w_j + ca_{j-1}\left(w_{j-1} + \delta w_{j-1}\right)\right] - s_j.$$

Since $ca_{j-1}w_{j-1} \subseteq s_j$, then

$$\delta s_j = \left(c\,\delta a_{j-1}w_j + ca_{j-1}\,\delta w_{j-1}\right) - s_j. \tag{I}$$

Equivalently, we can also derive

$$\delta s_j = \left(c\,\delta w_{j-1}a_j + cw_{j-1}\,\delta a_{j-1}\right) - s_j. \tag{II}$$

In general, when the rule contains ≥ 1 recursive predicates in the conjunction, the following theorem generalizes Equations (I) and (II) above.

Theorem. If

$$s \leftarrow c \wedge a^1 \wedge a^2 \wedge \cdots \wedge a^n,$$

where c is a derived predicate and the a^i, $i = 1, \ldots, n$, are recursively defined predicates, then

$$s_j + \delta s_j = s_j + c\left[\delta a^1_{j-1}\,a^2_j \cdots a^n_j + a^1_{j-1}\,\delta a^2_{j-1}\,a^3_j \cdots a^n_j + \cdots + a^1_{j-1}a^2_{j-1} \cdots \delta a^{n-1}_{j-1}\,a^n_j + a^1_{j-1}a^2_{j-1} \cdots a^{n-1}_{j-1}\,\delta a^n_{j-1}\right].$$

PROOF. The derivation steps used to derive Equation (I) are followed for the proof. □

The equation

$$\delta s_j = \left(c\left[\delta a_{j-1}^1 a_j^2 \cdots a_j^n + a_{j-1}^1 \delta a_{j-1}^2 a_j^3 \cdots a_j^n + \cdots + a_{j-1}^1 a_{j-1}^2 \cdots \delta a_{j-1}^{n-1} a_j^n + a_{j-1}^1 a_{j-1}^2 \cdots a_{j-1}^{n-1} \delta a_{j-1}^n\right]\right) - s_j \quad \text{(III)}$$

follows immediately from the theorem. The analysis culminating in Equation (III) clearly demonstrates that by using the differential approach only repeated computations are excluded in the join. Additionally, since we have proved the equivalence between both the naive and differential equations, the least fixed point semantics is preserved. It is important to note that, in computing δs_j using equation (III), only the δa_{j-1}^i and a_{j-1}^i, $i = 1, \ldots, n$, need be retained during the computation.

It is a pleasure to acknowledge the careful reading and comments of Michael Maher and Tsong Yueh Chen. Thanks also to the referees for their constructive remarks.

REFERENCES

1. Aho, A. V. and Ullman, J. D., University of Data Retrieval Languages, in *Proceedings of the 6th ACM Symposium on Principles of Programming Languages*, San Antonio, Tex., Jan. 1979, pp. 110–120.
2. Balbin, I. and Ramamohanarao, K., A Differential Approach to Query Optimisation in Recursive Deductive Databases, Technical Report 86/7, Dept. of Computer Science, Univ. of Melbourne, Apr. 1986.
3. Bancilhon, F., Naive Evaluation of Recursivcly Defined Relations, in *Proceedings of the Islamadora Conference on Database and AI*, 1985.
4. Bancilhon, F. and Ramakrishnan, R., An Amateur's Introduction to Recursive Query Processing Strategies, *Proceedings of ACM SIGMOD Conference*, 1986.
5. Bayer, R., Query Evaluation and Recursion in Deductive Database Systems, Technical Report, TUM-18503, Institut für Informatik, Technische Univ. München, Feb. 1985.
6. Henschen, L. J. and Naqvi, S. A., On Compiling Queries in Recursive First-Order Databases, *J. Assoc. Comput. Mach.* 31(1):47–85 (Jan. 1984).
7. McKay, D. P. and Shapiro, S. C., Using Active Connection Graphs for Reasoning with Recursive Rules, in *Proceedings of the Seventh IJCAI*, 1981, pp. 368–374.