EFFICIENT EVALUATION FOR A SUBSET OF RECURSIVE QUERIES

(Extended Abstract)

Gosta Grahne[1]
Seppo Sippu[2]
Eljas Soisalon-Soininen[1]

[1]Department of Computer Science
University of Helsinki
Tukholmankatu 2
SF-00250 Helsinki, Finland

[2]Department of Computer Science
University of Jyvaskyla
Seminaarinkatu 15
SF-40100 Jyvàskyla, Finland

Abstract Well-known results on graph traversal are used to develop a practical, efficient algorithm for evaluating regularly and linearly recursive queries in databases that contain only binary relations. Transformations are given that reduce a subset of regular and linear queries involving n-ary relations ($n > 2$) to queries involving only binary relations.

1 Introduction

Various strategies for processing logic queries in relational databases have been proposed (see the references in [4]) These strategies include general evaluation methods such as Naive Evaluation [5,13] and Semi-Naive Evaluation [2], Query/Subquery [17], Henschen-Naqvi [6], APEX [11], and the method used in Prolog implementations Another class of strategies, called query optimization strategies, try to transform the original query into a form that is more amenable to an underlying simple evaluation method such as Naive Evaluation These strategies include Aho-Ullman [1], Filtering [8,9], Magic Sets [3], Counting and Reverse Counting [3], and Generalized Counting [12]

A comparison of the strategies and their performance is given by Bancilhon and Ramakrishnan [4]. In a careful analysis of the evaluation of some typical binary queries Bancilhon and Ramakrishnan observed that the performance of a strategy is greatly influenced by the following three factors (1) the amount of duplication of work, (2) the size of the set of relevant facts, and (3) whether the intermediate results are represented as unary or binary relations Here duplication of work means the repeated firing of an inference rule on the same data This can happen in strategies that duplicate data (e g Prolog) and in strategies that do not remember previous firings (e g. Naive Evaluation and the iterative version of Query/Subquery)

The set of relevant facts is the set of tuples in the extensional database that need be consulted by a strategy to produce the answer to a given query The number of relevant facts tends to be large

The work was supported by the Alexander von Humboldt Foundation and by the Academy of Finland
Present address of E Soisalon-Soininen: Institute fur Angewandte Informatik und Formale Beschreibungsverfahren, Universitat Karlsruhe, D-7500 Karlsruhe, BRD

in bottom-up methods (Naive and Semi-Naive) Therefore these methods are usually coupled with some query optimization strategy that tries to reduce the number of relevant facts. This may be done by selection transposition (e g Aho-Ullman) or by introducing some additional, restricting inference rules (e g. Magic Sets).

Each of the general evaluation methods carries along a vector of intermediate relations that represent the current state of the evaluation In most methods these intermediate relations are of the same arity as those in the original database, whereas e.g the method of Henschen-Naqvi employs unary relations in the evaluation of binary relations Bancilhon and Ramakrishnan state that (in the case of binary queries) "strategies which only look at sets of nodes rather than sets of arcs perform better than those that look at sets of arcs, by an order of magnitude or more."

Most of the strategies proposed strive to capture the general case in which no restrictions are imposed on the form of the inference rules any kind of Horn clauses (without function symbols) and any kind of bindings of variables are allowed. Some strategies (e g Henschen-Naqvi) however may not permit recursion that is more complicated than linear. None of the strategies imposes restrictions on the arity of the relations: relations with arity ranging from unary to n-ary, $n > 1$, are allowed

Binary relations form an important subcase of n-ary relations. This is not only because of the fact that any set of relations can be represented as a set of binary relations (in fact many of the interesting examples of recursive queries, e g. "ancestor" and "cousins of the same generation", are binary) Problems on binary relations can usually be expressed as graph traversal problems For example, Hunt, Szymanski and Ullman [7] have shown that the problem of computing the value of any expression having binary relations as arguments and operators chosen from among $\cup$ (union), $\cdot$ (composition), $*$ (reflexive transitive closure), and $^{-1}$ (inverse) reduces to depth-first traversal of a certain directed graph constructed from this expression. Graph traversal is well understood, and there are very efficient general algorithms as well as algorithms that take into account the expected structure of the relation. For example, by applying Tarjan's strong components algorithm [16] to the graph constructed from an expression E with arguments of size n, we may compute the value of the expression in time $O(t \cdot n)$, where $t = \min\{|\mathrm{domain}(E)|, |\mathrm{range}(E)|\}$ [14]

In this paper we shall investigate in detail the complexity of evaluating regularly and linearly recursive queries when the relations in the database are binary relations. We shall generalize the algorithm of Hunt, Szymanski and Ullman to cover the linear case. The resulting algorithm will be "dynamic" in that the graph for the expression to be evaluated will be constructed incrementally as the traversing proceeds Finally, we shall show how a subset of regular and linear queries involving n-ary relations, $n > 2$, can be transformed into queries involving only binary relations

Based on graph traversal, our strategy for query evaluation is guaranteed to be efficient. This is seen immediately if we consider the three performance factors listed above. First, no duplication of work can occur because the graph is traversed only once (We shall take care that in the construction of the graph no data will be duplicated.) Second, the set of relevant facts is restricted to the set of reachable nodes. Third, the representation of intermediate results is extremely simple. At any moment of the evaluation, the portion of the graph constructed so far will represent the current state of the evaluation. Moreover, usually only the nodes, not arcs, of this graph need be stored Maintaining a set of nodes is of course easier and more efficient than maintaining collections of relations of different arity

For notation and definitions pertaining to function-free Horn clause programs we refer to [4].

2 Evaluation of binary relations

We assume that the intensional database consists of rules of the forms:

(A1) $p(X_1,X_{n+1}) :- p_1(X_1,X_2),p_2(X_2,X_3),\ldots,p_n(X_n,X_{n+1}).$

(A2) $p(X,Y) - s(Y,X)$

In (A1) $X_1,\ldots,X_{n+1}$ ($n \geq 0$) are distinct variables, $p_1,\ldots,p_n$ are base relations, derived relations, or evaluable predicates In (A2) X and Y are distinct variables and s is a base relation, a derived relation, or an evaluable predicate.

The evaluation algorithm will require that the base relations and evaluable predicates appearing in the rules are range-restricted. More spacifically, given any base relation or evaluable predicate r and any term u, it should be possible to determine effectively the set of all terms v satisfying r(u,v) and the set of all terms w satisfying r(w,u).

Lemma 1 Any set of linear [4] rules of the forms (A1) and (A2) can be transformed into a set of equations of the form

(A') $p = e_p$

such that the following conditions are satisfied

(1) There is exactly one equation for each derived relation p

(2) The right-hand side e_p of the equation for p is an expression whose arguments are base relations, derived relations, or evaluable predicates and whose operators are chosen from among $\cup$ (union), $\cdot$ (composition), * (reflexive transitive closure), and $^{-1}$ (inverse)

(3) If e_p contains a subexpression of the form f^{-1}, then f is a relation, not a more complicated expression (cf. [15], definition 3).

(4) e_p contains at most one occurrence of a derived relation.

(5) If p is a linear relation, then p occurs in e_p.

(6) If r is a derived relation occurring in e_p, then r is linear.
□

For example, the system of linear rules

```
p(X,Z) - r(X,Y),b1(Y,Z).
r(X,Y):- s(X,Y).
r(X,Z):- b2(X,Y),p(Y,Z).
s(X,Y).- b3(X,Y).
s(X,Z) - s(X,Y),b4(Y,Z)
p1(X,Y).- s(Y,X).
```
$p1(X_1,X_4) :- s(X_1,X_2),p(X_2,X_3),b1(X_3,X_4)$

can be transformed into the set of equations

$p = (b3\cdot b4^* \cup b2\cdot p)\cdot b1,$
$r = b3\cdot b4^* \cup b2\cdot r\cdot b1,$
$s = b3\cdot b4^*,$
$p1 = (b4^{-1})^*\cdot b3^{-1} \cup b3\cdot b4^*\cdot p\cdot b1.$

We shall represent an equation $p = e_p$ as a nondeterministic finite automaton, denoted by $M(e_p)$. For expression e, M(e) is the automaton obtained by the standard technique from e when we regard e as a regular expression over the alphabet

$\{r \mid r \text{ is a relation appearing in } e\} \cup \{r^{-1} \mid r \text{ is a relation appearing in } e\}$

(Cf. [15].)

The evaluation of a query for p will be controlled by a hierarchy of automata denoted by EM(p,i), $i \geq 1$ The i^{th} iteration of the main loop of the algorithm will be controlled by EM(p,i) EM(p,1) is a copy of $M(e_p)$ If $i > 1$ and EM(p,i-1) contains a transition $q \xrightarrow{r} q'$ where r is a derived relation (usually r = p), then EM(p,i) is obtained from EM(p,i-1) by replacing this transition by a fresh copy of $M(e_r)$ More specifically, the transition $q \xrightarrow{r} q'$ is removed and transitions $q \xrightarrow{\varepsilon} q_s'$ and $q_f' \xrightarrow{\varepsilon} q'$ are added, where ε is the empty string and q_s' and q_f' are the initial and final states of the copy of $M(e_r)$ (see Fig 1) If EM(p,i-1) contains a transition $q \xrightarrow{r^{-1}} q'$ where r is a derived relation, then EM(p,i) is obtained by replacing this transition by a fresh copy of the inverse of $M(e_r)$. The inverse of automaton M(e) is the nondeterministic finite automaton obtained from M(e) by exchanging the initial and final states and by replacing (1) each transition $q \xrightarrow{s} q'$ by the transition $q' \xrightarrow{s^{-1}} q$, (2) each transition $q \xrightarrow{s^{-1}} q'$ by the transition $q' \xrightarrow{s} q$, and (3) each transition $q \xrightarrow{\varepsilon} q'$ by the transition $q' \xrightarrow{\varepsilon} q$. (Here s is any relation.)

An interpretation of EM(p,i) is a directed graph obtained from EM(p,i) by replacing each transition $q \xrightarrow{r} q'$, where r is a base relation or an evaluable predicate (or the inverse of such), by zero or more arcs of the form ((q,u),(q',v)), where r(u,v) is true (Cf. [15,10].)

Now consider a query of the form

query(Y) - p(V,Y)

Here V is a subset of the domain of p. The evaluation algorithm will generate a sequence of interpretations of EM(p,i), i = 1,2, ,h, where h is a certain upper bound (to be discussed later). The interpretation of EM(p,i) is denoted by G(p,V,i) (see Fig 2).

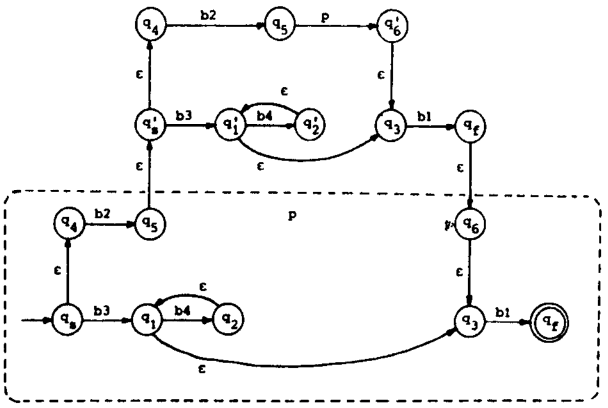


Figure 1 Automaton EM(p,2) used in the second iteration of the main loop for evaluating $p = e_p$ when e_p is the expression (b3 b4* ∪ b2 p) b1 (Here b1, b2 b3 and b4 are base relations or evaluable predicates) The portion enclosed by a broken line shows the automaton EM(p 1) ($=M(e_p)$) with transition $q_5 \xrightarrow{p} q_6$ removed

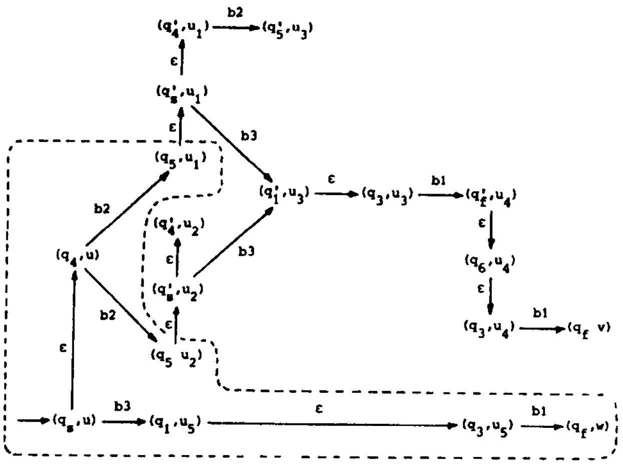


Figure 2 Graph G(p,V 2) when p is as in Fig 1, V = {u}, and the extensional database contains the facts $b1(u_3\ u_4)$, $b1(u_4\ v)$, $b1(u_5\ w)$ $b2(u,u_1)$, $b2(u,u_2)$ $b2(u_1\ u_3)$, $b3(u_1\ u_3)$, $b3(u_2,u_3)$, $b3(u\ u_5)$ $b3(u_3\ u_3)$ The portion enclosed by a broken line shows G(p,V,1)

The algorithm starts with G(p,V,0), which is the graph with set of nodes $\{(q_s,u) \mid u \in V\}$ and with no arcs (q_s is the initial state of all EM(p,i), $i \geq 1$) During the i^{th} iteration ($i \geq 1$) of the main loop, G(p,V,i-1) will be extended to G(p,V,i). This is done by performing a depth-first traversal. When i = 1 the traversal starts from all nodes (q_s,u), $u \in V$

All paths not containing arcs labelled with derived relations are traversed Whenever a node (q,u) not visited before is entered, all transitions in $EM(p,i)$ leaving q are examined For any transition $q \xrightarrow{\varepsilon} q'$ such that (q',u) has not yet been generated, the algorithm generates (q',u) and continues the traversal from this node For any transition $q \xrightarrow{r} q'$ where r is a base relation or an evaluable predicate and for any term v such that $r(u,v)$ is true and the node (q',v) has not yet been generated, the algorithm generates (q',v) and continues the traversal from this node For any transition $q \xrightarrow{r^{-1}} q'$ where r is a base relation or an evaluable predicate and for any term v such that $r(v,u)$ is true and the node (q',v) has not yet been generated, the algorithm generates (q',v) and continues the traversal from this node

At the end of the ith iteration, it is examined whether or not a new iteration, the (i+1)th, is needed. If yes, $EM(p,i)$ is expanded into $EM(p,i+1)$, and a new depth-first traversal is performed. The traversal now starts from all nodes (q_s',u), where q_s' is the initial state of the newly added copy of automaton $M(e_r)$ (usually $= M(e_p)$) (or its inverse) and $q \xrightarrow{\varepsilon} q_s'$ is a newly added transition such that $G(p,V,i)$ contains (q,u). If on the contrary, the algorithm decides to stop after the ith iteration, the answer to the query can be read from the nodes (q_f,u), where q_f is the final state of $EM(p,i)$. The answer set Y is $\{u \mid (q_f,u) \in G(p,V,i)\}$.

If e_p contains no occurrence of a derived relation (the regular case), then only a single iteration of the main loop is needed. Only the automaton $M(e_p)$ and the graph $G(p,V,1)$ need be constructed, and the answer to the query will be $Y = \{u \mid (q_f,u) \in G(p,V,1)\}$ In fact, the graph $G(p,V,1)$ then consists exactly of the reachable portions of the graph for e_p considered by Hunt, Szymanski and Ullman [7] Here "reachable" means "reachable from some node in $\{(q_s,u) \mid u \in V\}$".

For the linear case we have to derive some upper bound on the number of iterations. First we note that after expanding $EM(p,i)$ into $EM(p,i+1)$ it may turn out that there are no nodes (q_s',u) from which to start a new traversal This happens when in the previous traversal no node (q,u) is visited where q has a transition on a derived relation In this case the algorithm naturally must stop because further iterations cannot extend the answer set

To handle the general case the algorithm maintains a set, D, that at any moment contains those terms in the domain of the linear relation that have been reached so far The algorithm will stop when more than $|D|$ iterations of the main loop have been executed since the latest generation of a new answer node (q_f,u).

<u>Lemma 2</u>. The time taken by the algorithm to answer to the query $p(V,Y)$ is

$$O(|G(p,V,h)| \log(facts+|G(p,V,h)|)),$$

where h is the number of iterations executed, facts is the number of tuples in the base relations consulted, and $|G(p,V,h)|$ is the number of nodes in $G(p,V,h)$ □

Here we have assumed that the base relations, and the graph $G(p,V,h)$, are stored using a data structure from which data items can be retrieved in log time Observe that only the nodes, not arcs, of $G(p,V,h)$ need be stored

<u>Theorem 3</u>. (The regular case) If e_p does not contain any occurrence of a derived relation, the time needed to answer to the query $p(V,Y)$ is $O(n \log n)$, where n is the number of tuples in the base relations (and the evaluable predicates) appearing in e_p. □

Theorem 4 (The linear case) Assume that the equation for p is

$$(L) \quad p = e_0 \cup e_1\, p\, e_2,$$

where e_0, e_1, and e_2 do not contain occurrences of derived relations. Denote by E_i ($i \geq 0$) the expression defined by:

$$E_i = \begin{cases} e_0, & \text{when } i = 0; \\ (e_0 \cup e_1\, E_{i-1} \cdot e_2), & \text{when } i > 0. \end{cases}$$

The time taken by the algorithm to answer to the query p(V,Y) is the same as the time taken to evaluate the same query in the regular case $p = E_h$, where h is the number of iterations needed in the linear case (L). Hence the time needed to answer to the query p(V,Y) is $O(h\, n \log(h \cdot n))$, where n is the number of tuples in the base relations (and evaluable predicates) appearing in $e_0 \cup e_1 \cup e_2$ and (1) h is the length of the longest path in $e_1|V$ when $e_1|V$ is acyclic and (2) h is $|domain(p|V)|\ |range(p|V)|$ when $e_1|V$ is cyclic. (Here $r|V$ denotes those portions of r that are reached from nodes in V) □

Observe that the expression E_h is equivalent to the expression

$$E'_h = e_0 \cup e_1\, e_0 \cdot e_2 \cup e_1^2 \cdot e_0\, e_2^2 \cup \ . \ \cup e_1^h\, e_0\, e_2^h$$

in that it denotes exactly the same relation However, E_h is essentially (by a factor of h) smaller than E'_h.

As an example assume that the rules for derived relations are

```
sg(X,X)
sg(X,Y) - parent(X,X'),sg(X',Y'),
                child(Y',Y).
child(X,Y):- parent(Y,X).
```

The time needed to determine the set of all persons Y such that john and Y are cousins at the same generation, is

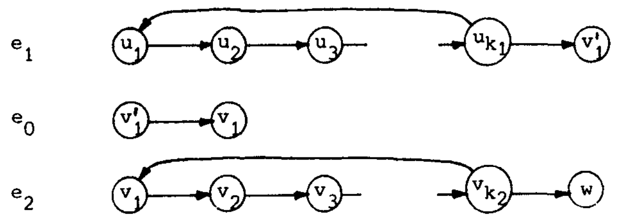
$$O((k_0+ \quad +k_h) \log(|people| + |parent| + k_0 + .. + k_h)),$$

where h is the number of generations from john to his remotest ancestor and k_i is the number of persons v satisfying

$$john \ \ parent^i\ child^j \ \ v,$$

for some j, $0 \leq j \leq i$. We cannot imagine a solution more efficient than this!

To exemplify the worst case let k_1 and k_2 be distinct prime numbers and let e_0, e_1, and e_2 be the following relations



Now $O(k_1 k_2)$ iterations of the main loop of the algorithm are needed to produce the entire answer to the query $p(\{u_1\},Y)$, when $p = e_0 \cup e_1 \cdot p\, e_2$. This is because (u_1,w) belongs to the relation denoted by

$$e_1^{k_1 k_2} \cdot e_0 \cdot e_2^{k_1 k_2}$$

but does not belong to any $e_1^k\, e_0 \cdot e_2^k$, where $k < k_1 k_2$. Observe that the algorithm performs periodically k_1 successive iterations during which nothing new is added to the answer set. (This is an example of a case in which a bottom-up evaluation method, such as Naive Evaluation, shows its best!)

We conclude this section by noting that the algorithm outlined above could be used as such to evaluate the query p(X,Y), where the entire relation p, not only an image p(V), is wanted. We simply execute the algorithm for all V = {u}, where u is a term in the domain of p. This yields the time bound $O(|domain(p)|\ n \cdot \log n)$ in the regular case However, the graphs

G(p {u},i) may intersect for different u's, which means duplication of work. This duplication can be avoided by applying Tarjan's strong components algorithm [16] (cf. [14])

3 Transforming n-ary queries into binary queries

A typical set of rules for a regular n-ary relation p can be represented as

$$p(\bar{X}_1,\bar{X}_2) :- r0(\bar{X}_1,\bar{X}_2)$$

$$\text{(R)}\quad \begin{aligned} p(\bar{X}_1,\bar{X}_2) :- \ & r1(\bar{Y}_1,\bar{Y}_2),\ p(\bar{Z}_1,\bar{Z}_2),\\ & pr1(\bar{X}_1,\bar{Y}_1),\\ & r1p(\bar{Y}_2,\bar{Z}_1),\\ & pp(\bar{Z}_2,\bar{X}_2) \end{aligned}$$

Here $\bar{X}_1$, $\bar{X}_2$, $\bar{Y}_1$, $\bar{Y}_2$, $\bar{Z}_1$, and $\bar{Z}_2$ are vectors of distinct variables No variable has two occurrences in a vector, and no two vectors have common variables The length of the concatenated vector $\bar{X}_1,\bar{X}_2$ is n. The vectors $\bar{X}_2$ and $\bar{Z}_2$ have equal length. The relations r0, r1, pr1, r1p, and pp are nonrecursive relations (base relations, evaluable predicates, or derived relations). r0 and r1 are range-restricted To guarantee that the system indeed is regular, we require that the relation pp satisfy the following "transitivity" condition

$pp(\bar{z},\bar{x})$ whenever $\bar{x}$, $\bar{y}$, and $\bar{z}$ are equal-length vectors of terms and $pp(\bar{z},\bar{y})$ and $pp(\bar{y},\bar{x})$.

The pair of rules (R) captures the essence of a regular n-ary relation. We argue that for a large class of regular relations the rules can be automatically transformed into form (R)

As a simple example, consider a database representing airline flights [1] The extensional database consists of facts of the form

flight(s,dt,d,at),

where s and d are the source and the destination of a flight and dt and at are the departure and arrival times. The problem is to evaluate the derived relation transitflight defined by:

```
transitflight(S,DT,D,AT) -
    flight(S,DT,D,AT)
transitflight(S,DT,D,AT) -
    flight(S,DT,D1,AT1),
    transitflight(D1,DT1,D,AT),
    AT1 < DT1
```

The pair of rules is equivalent to the following set of rules.

```
transitflight(S,DT,D,AT).-
    flight(S,DT,D,AT)
transitflight(S,DT,D,AT) -
    flight(S1,DT1,D1,AT1),
    transitflight(S2,DT2,D2,AT2),
        pr1(S,DT,S1,DT1),
        r1p(D1,AT1,S2,DT2),
        pp(D2,AT2,D,AT).
pr1(S,DT,S1,DT1).- S = S1, DT = DT1
r1p(D1,AT1,S2,DT2)·- D1 = S2, AT1 < DT2
pp(D2,AT2,D,AT):- D2 = D, AT2 = AT
```

This set of rules is of the required form (R) Here r0 = flight = r1. Also note that pp is trivially transitive

To make possible the use of the evaluation algorithm presented in the previous section we have to shift from n-ary relations to binary relations. For any pair of rules of the form (R) we define binary relations r1r1b, r1r0b, and pr1b by.

$$r1r1b(tail(\bar{Y}_2),tail(\bar{U}_2)) - r1(\bar{U}_1,\bar{U}_2),\ r1p(\bar{Y}_2,\bar{Z}_1),\ pr1(\bar{Z}_1,\bar{U}_1)$$

$$r1r0b(tail(\bar{Y}_2),tail(\bar{Z}_2)) - r0(\bar{Z}_1,\bar{Z}_2),\ r1p(\bar{Y}_2,\bar{Z}_1).$$

$$pr1b(head(\bar{X}_1),tail(\bar{Y}_2)) - r1(\bar{Y}_1,\bar{Y}_2),\ pr1(\bar{X}_1,\bar{Y}_1).$$

Here $\bar{X}_1$, $\bar{Y}_1$, $\bar{Y}_2$ $\bar{Z}_1$, $\bar{Z}_2$, $\bar{U}_1$, $\bar{U}_2$ are vectors of distinct variables. The vectors $\bar{Y}_2$ and $\bar{U}_2$ are of the same length as the vector $\bar{Y}_2$ in (R). The vector $\bar{Z}_2$ is of the same length as the vector $\bar{Z}_2$ in (R)

For example, in the "flight" database we have:

r1r1b(tail(d,at),tail(d1,at1)) if and only if flight(d,dt,d1,at1) and at < dt for some dt.

r1r0b = r1r1b.

pr1b(head(s,dt),tail(d,at)) if and only if flight(s,dt,d,at)

We have used compound terms with function symbols (head, tail) to group together attribute values in the original tuples. Observe that r1r1b, r1r0b, and pr1b are all range-restricted because r0 and r1 are so. Hence from the point of view of the evaluation algorithm we may regard r1r1b, r1r0b, and pr1b as evaluable predicates. For example, given a compound term head($\bar{x}_1$) the evaluation of pr1b(head($\bar{x}_1$),tail($\bar{Y}_2$)) will generate the set of all terms tail($\bar{y}_2$) such that for some vector $\bar{y}_1$ the clauses r1($\bar{y}_1$,$\bar{y}_2$) and pr1($\bar{x}_1$,$\bar{y}_1$) are true. In this generation, the standard retrieval mechanism of the extensional database is used (together with a simple inference mechanism involving only nonrecursive predicates)

<u>Theorem 5</u> Let

pb = pr1b·r1r1b*·r1r0b.

Then p can be evaluated using the nonrecursive rules

p($\bar{X}_1$,$\bar{X}_2$) - r0($\bar{X}_1$,$\bar{X}_2$)

p($\bar{X}_1$,$\bar{X}_2$) - pb(head($\bar{X}_1$),tail($\bar{Z}_2$)),
pp($\bar{Z}_2$,$\bar{X}_2$)

□

Now if M is the number of tuples in the relations pr1b, r1r1b, and r1r0b, we conclude from Theorem 3 that any query of the form

query($\bar{X}_2$):- p($\bar{v}_1$,$\bar{X}_2$)

can be evaluated in time O(M log M) In the worst case M may be quadratic in N, the number of tuples in the original relations r0 and r1. Hence the worst case time bound for the query is O(N^2 log N).

However, in most cases we can tighten this bound by a factor of N. This is possible when r0 and r1 are base relations and when the data structure used to implement these relations implies a linear order on the tuples and the retrieval mechanism allows tuples to be retrieved efficiently in this order. More specifically, we assume that for base relation r the following predicates are efficiently computable.

rfirstaddr(A).- "A is the smallest address of a tuple in r".

rnextaddr(A,B) - "B is the smallest address of a tuple in r satisfying A < B".

rtuple(A,$\bar{Y}$):- "$\bar{Y}$ is the tuple of r at address A".

Now define:

r1firstb(tail($\bar{Y}_2$),t($\bar{Y}_2$,A)):-
r1firstaddr(A).

r1nextb(t($\bar{Y}_2$,A),t($\bar{Y}_2$,B)):-
r1nextaddr(A,B).

r1r1downb(t($\bar{Y}_2$,A),tail($\bar{U}_2$)):-
r1tuple(A,$\bar{U}_1$,$\bar{U}_2$), r1p($\bar{Y}_2$,$\bar{Z}_1$),
pr1($\bar{Z}_1$,$\bar{U}_1$).

Clearly, r1firstb, r1nextb, and r1r1downb are all of linear size and

r1r1b = r1firstb r1nextb*·r1r1downb.

Similarly, we may define linear-size relations r0firstb, r0nextb, r1r0downb, and pr1downb satisfying

r1r0b = r0firstb r0nextb* r1r0downb,

pr1b = r1firstb·r1nextb* pr1downb.

Now pb can be expressed as

pb = r1firstb r1nextb*·pr1downb
(r1firstb r1nextb* r1r1downb)*
r0firstb r0nextb* r1r0downb

All arguments in this expression are of size linear in the size of the original relations r0 and r1 and hence we get the time bound O(N log N)

The above ideas can easily be generalized to the linear case. A typical set of rules for a linear n-ary relation can be represented as

$p(\bar{X}_1,\bar{X}_2)$ - $r0(\bar{X}_1,\bar{X}_2)$

$p(\bar{X}_1,\bar{X}_2)$:-

(L) $r1(\bar{Y}_1,\bar{Y}_2)$, $p(\bar{Z}_1,\bar{Z}_2)$, $r2(\bar{W}_1,\bar{W}_2)$,
$pr1(\bar{X}_1,\bar{Y}_1)$,
$r1p(\bar{Y}_2,\bar{Z}_1)$,
$pr2(\bar{Z}_2,\bar{W}_1)$,
$r2p(\bar{W}_2,\bar{X}_2)$.

Here $\bar{X}_1$, $\bar{X}_2$, $\bar{Y}_1$, $\bar{Y}_2$, $\bar{Z}_1$, $\bar{Z}_2$, $\bar{W}_1$, and $\bar{W}_2$ are vectors of distinct variables. No variable has two occurrences in a vector, and no two vectors have common variables. The length of the concatenated vector $\bar{X}_1,\bar{X}_2$ is n The vectors $\bar{X}_2$ and $\bar{Z}_2$ have equal length. The relations r0, r1, r2, pr1, r1p, pr2, and r2p are nonrecursive relations. r0, r1, and r2 are range-restricted

For example, the pair of rules

sg(X,X)

$sg(X_1,X_2)$ - $parent(X_1,X_1')$, $sg(X_1',X_2')$,
$parent(X_2,X_2')$.

is equivalent to.

$sg(X_1,X_2)$ - $equal(X_1,X_2)$

$sg(X_1,X_2)$ - $parent(Y_1,Y_2)$, $sg(Z_1,Z_2)$,
$child(W_1,W_2)$,
$equal(X_1,Y_1)$,
$equal(Y_2,Z_1)$,
$equal(Z_2,W_1)$,
$equal(W_2,X_2)$.

equal(X,Y) - X = Y.

child(X,Y).- parent(Y,X).

<u>Theorem 6</u>. Let the rules for p be of the form (L) Then there are range-restricted binary relations pr1b, r1r1b, r1r0b, r0r2b, and r2r2b such that when

pb = pr1b·pb', and

pb' = r1r0b r0r2b U r1r1b·pb'·r2r2b,

then p can be evaluated using the non-recursive rules

$p(\bar{X}_1,\bar{X}_2)$ $r0(\bar{X}_1,\bar{X}_2)$.

$p(\bar{X}_1,X_2)$ ·- $pb(head(\bar{X}_1),tail(\bar{W}_2))$,
$r2p(\bar{W}_2,\bar{X}_2)$.

□

As in the regular case we may represent the relations pr1b, r1r1b, r1r0b, r0r2b, and r2r2b as expressions containing only linear-size arguments This is possible when r0, r1, and r2 are base relations and the predicates rfirstaddr, rnextaddr, and rtuple are available for these relations.

4 Conclusion

We have presented an efficient strategy for evaluating a subset of regularly and linearly recursive queries The strategy is based on a graph traversal algorithm that can solve linearly recursive equations involving binary relations and the operations ∪ (union), · (composition), * (reflexive transitive closure), and $^{-1}$ (inverse). The algorithm is a generalization of an algorithm originally presented by Hunt, Szymanski and Ullman [7] for evaluating binary relational expressions.

We believe that our strategy applies to a fairly large set of recursive queries encountered in practice. However, the strategy has its limitations, first of all because the underlying graph traversal algorithm only allows for binary relations the set of operations $\{\cup, \cdot, *, ^{-1}\}$. An interesting question is whether or not this set can be extended without compromising the efficiency stemming from graph traversal. Another important topic of further research is to develop algorithms that, given an arbitrary query, can detect whether or not this query can be evaluated using our strategy, that is, whether or not the query can be transformed into one of the forms (R) or (L) considered in Section 3.

Acknowledgement

The authors are indebted to Mr Juhani Kuittinen for fruitful discussions on the topics of Section 2. Kuittinen has implemented a system for evaluating binary relational expressions [10]

References

1 A.Aho and J.Ullman. Universality of data retrieval languages. Proc. 6th ACM Symp. on Principles of Programming Languages.

2 F Bancilhon: Naive evaluation of recursively defined relations. In: On Knowledge Base Management Systems - Integrating Database and AI Systems. Springer-Verlag, 1985.

3. F.Bancilhon, D.Maier, Y.Sagiv and J. Ullman: Magic sets and other strange ways to implement logic programs. Proc 5th ACM SIGMOD-SIGACT Symp. on Principles of Database Systems, 1986.

4 F Bancilhon and R.Ramakrishnan: An amateur's introduction to recursive query processing strategies Proc. ACM SIGMOD'86, SIGMOD Record (ACM) 15 2 (1986)

5. C.Chang: On the evaluation of queries containing derived relations in relational databases. In: Advances in Data Base Theory, Vol 1 Plenum Press, 1981.

6. L.Henschen and S Naqvi. On compiling queries in recursive first-order data bases. J. ACM 31:1 (1984)

7. H.Hunt, T.Szymanski and J.Ullman: Operations on sparse relations. Comm. ACM 20 3 (1977).

8. M Kifer and E.Lozinskii. A framework for an efficient implementation of deductive database systems. Proc. 6th Advanced Database Symp., Tokyo, 1986.

9. M.Kifer and E.Lozinskii: Filtering data flow in deductive databases. Internat. Conf. on Database Theory, Rome, 1986.

10. J.Kuittinen: Binary relations and relational expressions (in Finnish). Internal Report, Dept. of Computer Sci., Univ. of Helsinki, 1986.

11. E.Lozinskii: Evaluating queries in deductive databases by generating. Proc. 11th Internat. Joint Conf. on Artificial Intelligence, 1985.

12. D.Saccà and C Zaniolo: The generalized counting method for recursive logic queries Internat. Conf. on Database Theory, Rome, 1986.

13. S.Shapiro and D.McKay. Inference with recursive rules Proc 1st Annual National Conf on Artificial Intelligence, 1980.

14. S.Sippu and E.Soisalon-Soininen: On the use of relational expressions in the design of efficient algorithms. Automata, Languages and Programming, 12th Colloquium. Springer-Verlag, 1985.

15. T.Szymanski and J.Ullman: Evaluating relational expressions with dense and sparse arguments. SIAM J. Comput. 6·1 (1977).

16. R Tarjan: Depth-first search and linear graph algorithms. SIAM J. Comput. 1·2 (1972).

17. L.Vieille: Recursive axioms in deductive databases· the Query/Subquery approach. Proc. 1st Internat. Conf. on Expert Database Systems, 1986.