# AN ALGEBRAIC APPROACH TO RECURSIVE INFERENCE

by

Yannis E. Ioannidis and Eugene Wong

Memorandum No. UCB/ERL M85/93

5 December 1985

# AN ALGEBRAIC APPROACH TO RECURSIVE INFERENCE

by

Yannis E. Ioannidis and Eugene Wong

Memorandum No. UCB/ERL M85/93

5 December 1985

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

# AN ALGEBRAIC APPROACH TO RECURSIVE INFERENCE

**Yannis E. Ioannidis**
**Eugene Wong**

*Department of Electrical Engineering and Computer Science*
*Computer Science Division*
*University of California*
*Berkeley, CA 94720*

**Abstract**

Recursion in the database context has traditionally been studied under the formalism of 1-st order logic. In particular, the bulk of the research effort in the last few years has been devoted to recursive Horn clauses. In this paper we reformulate the recursion problem in operator form. The relational operators are embedded in a partially ordered semiring with identity. This algebraic structure so obtained enables us to get more information about the mechanics of recursion. One result of this is that we have been able to obtain a significant decomposition theorem for recursive queries.

This research was supported by the National Science Foundation under Grant ECS-8300463

## 1. INTRODUCTION

Consider a function-free, constant-free Horn clause [Gall78]

$$P_0(\underline{x}^{(0)}) \wedge Q_1(\underline{x}^{(1)}) \wedge \cdots \wedge Q_k(\underline{x}^{(k)}) \rightarrow P_{k+1}(\underline{x}^{(k+1)}) \tag{1}$$

where for each $i$, $\underline{x}^{(i)}$ is a subset of some fixed set of variables $(x_1,x_2,...,x_n)$. We say the formula is *recursive* if $P_{k+1} = P_0 = P$ (but $\underline{x}^{(0)} \neq \underline{x}^{(k+1)}$). For the purposes of this paper we further assume that $Q_i = P$ for no other $i$. In this case we say that the formula is *linear recursive.* A familiar example is the following:

$$ancestor(X,Z) \wedge father(Z,Y) \rightarrow ancestor(X,Y)$$

where all variables are assumed universally quantified.

A recursive Horn clause can be interpreted in terms of relations as follows: Let $P$, $\{Q_i\}$ be relations and $f(P, \{Q_i\})$ a function with values that are relations over the same columns as $P$. Then, (1) with $P_0 = P_{k+1} = P$ takes on the form

$$f(P,\{Q_i\}) \subseteq P$$

or equivalently

$$P \bigcup f(P,\{Q_i\}) = P \tag{2}$$

The problem of recursive inference can now be stated in relational form as follows:

Given relations $R,Q_1,...,Q_k$ and function $f$, find $P$ such that

(a) $P \bigcup f(P,\{Q_i\}) = P$

(b) $R \subseteq P$

(c) $P$ is minimal with respect to (a) and (b), i.e. $P'$ satisfying (a) and (b) implies $P \subseteq P'$.

It is clear that a solution to (a) and (b) exists, since the cartesian product of the domains of $P$ is such a solution, and that the intersection of all relations satisfying (a) and (b) is the unique minimal solution $P$ satisfying (a), (b) and (c). It is well known that $P$ can be found by iteration. If the initial value $R$ is a finite relation, then the iteration always terminates. In [Ioan85] it is shown that for some recursions the number of iterations is bounded by an integer independent of $R$ and

$\{Q_i\}$. Such recursions are said to be *uniformly bounded,* and [Ioan85] gives a graph characterization of all such uniformly bounded recursions.

In this paper we reformulate the recursion problem in operator form. Aside from some notational advantages, the algebraic structure of the operators (a partially ordered semiring with identity) can be exploited to yield results that may prove important in processing recursions. Although our study is only preliminary, a significant decomposition theorem has already been obtained.

This paper is organized as follows: In Section 2 we define the set of relational operators as a semiring and describe the recursion problem in operator form. Section 3 discusses the class of bounded recursions from the algebraic viewpoint. In Section 4 we introduce the transitive closure of an algebraic operator as a combination of a multiplicative and additive inverse. Section 5 contains the description of a graph model for recursion and presents some initial results in recursion decomposition using algebraic manipulation. Finally, in Section 6 we summarize our results and describe some possible directions of our future work.

## 2. SEMIRING OF RELATIONAL OPERATORS

For fixed relations $\{Q_i\}$, the function $f(P,\{Q_i\})$ is an operator on $P$ and can be written more clearly as $AP$, where $A$ is an operator mapping relations over a fixed set of domains into ones over the same domains. *Multiplication* of operators is defined by

$$(A * B)P = A(BP)$$

and *addition* by

$$(A+B)P = AP \bigcup BP$$

For notational convenience we omit the operator *. So, whenever we write $ABP$, with $A, B$ operators and $P$ a relation, we actually mean $(A * B)P$. Identity ($1P = P$) and null ($0P = \emptyset$) are defined in obvious ways.

Further, we can define the $n$-th power of an operator $A$ as:

$$A^n = A * A * \cdots * A$$

$$\longleftarrow n \text{ times} \longrightarrow$$

For $n=0$ we have $A^0 = 1$, where 1 is the identity operator.

We note that multiplication is associative, addition is symmetric, but neither has an inverse. Further, multiplication is distributive over addition. Thus, relational operators form a *Semiring* (but not a ring because there is no additive inverse).

A partial order can be defined on relational operators using set inclusion, i.e.,

$$A \leq B \iff \forall P,\ AP \subseteq BP$$

With respect to multiplication and addition the partial order enjoys the following properties:

(a) $A = A + A,\ A \leq A + B$

(b) $A \leq B \iff A + B = B$ (3)

(c) $A \leq B \implies A + C \leq B + C$

(d) $A \leq B$ , $C \leq D$ and $A, B$ are monotone $\implies A\ C \leq B\ D$

The proofs of these properties are straightforward and will be omitted. For (d), note that all relational operators under consideration are monotone, i.e. for any operator $A$ and any two relations $P, Q$ it is $P \subseteq Q \implies AP \subseteq AQ$.

The problem of recursion can now be restated as follows: Given operator $A$, find $B$ satisfying:

(a) $(1+A)B = B$

(b) $B \geq 1$ (4)

(c) $B$ is minimal with respect to (a) and (b), i.e. for all other $C$ satisfying (a) and (b), it is $B \leq C$.

The solution can again be obtained by iteration:

$B_0 = 1$

$B_{n+1} = (1+A)B_n = (1+A)^{n+1}$

From (3a) we have $A^k + A^k = A^k$ for every $k$, so it is

$$(1+A)^n = \sum_{k=0}^{n} A^k$$

For every finite relation $R$, there exists $n_0$ (depending on $R$) such that

$$(1+A)^n R = (1+A)^{n_0} R, \quad \forall \ n \geq n_0$$

Hence, $\lim_{n\to\infty} B_n$ is well defined as a pointwise limit, and is equal to

$$A^* = \sum_{k=0}^{\infty} A^k \tag{5}$$

which is a complete solution to the recursion problem. The operator $A^*$ will be called the *transitive closure* of $A$.

What remains for us to prove is that $A^*$ is indeed the minimal solution (least fixpoint) of $(1+A)B = B$ with $B \geq 1$. That is, we will prove that for all operators $B$ that satisfy (4a) and (4b) it is $B \geq A^*$. We will do it by induction on the number of terms in $A^* = \sum_{k=0}^{\infty} A^k$.

*Basis:* For $n=0$ it is $\sum_{k=0}^{0} A^k = 1$ and we have from (4b) that $B \geq 1$.

*Induction Step:* Assume that $B \geq \sum_{k=0}^{n} A^k$ for some $n \geq 0$. From this we have

$$B \geq \sum_{k=0}^{n} A^k \implies (1+A)B \geq (1+A) \sum_{k=0}^{n} A^k, \qquad \text{since } A \text{ is monotone}$$

$$\implies (1+A)B \geq \sum_{k=0}^{n+1} A^k, \qquad \text{from (3)}$$

$$\implies B \geq \sum_{k=0}^{n+1} A^k, \qquad \text{from (4)}$$

So, for all $n \geq 0$ we have that $B \geq \sum_{k=0}^{n} A^k$. Since the sequence (of the partial sums) is upwards bounded by $B$ and it is monotone, we can conclude that its limit $A^*$ is also bounded by $B$. Hence, for any $B$ satisfying $B = (1+A)B$ and $B \geq 1$ it is $B \geq A^*$. This implies that $A^*$ is

the least such operator, i.e. it is the least fixpoint of (4).

## 3. BOUNDED RECURSION

Consider the problem of recursion in the relational form (2) and define the iterative solutions by

$$P_{n+1} = P_n \bigcup f(P_n, \{Q_i\})$$

The corresponding recursion is said to be *bounded* by $N$ if

$$P_{N+1} = P_N$$

whence it follows that $P_{N+k} = P_N$ for all $k \geq 0$. This bound $N$ depends on $P_0$ and $\{Q_i\}$ in general.

In the context of operators all relations in $\{Q_i\}$ are fixed. Hence, this bound varies with $P_0$ only. In the case that the recursion bound is independent of $P_0$ the underlying operator $A$ is such that

$$(1+A)^{N+k} = (1+A)^N \text{ for all } k \geq 0,$$

in which case the closure is given by

$$A^* = (1+A)^N = \sum_{k=0}^{N} A^k \tag{6}$$

An operator $A$ satisfying (6) is said to be *N-reducible.*

**Proposition 3.1:** The following conditions are equivalent:

(i) $A$ is $N$-reducible.

(ii) $A^{N+1} \leq (1+A)^N$.

(iii) $(1+A)^N(1+A)^N = (1+A)^N$ , i.e. $(1+A)^N$ is idempotent.

**Proof:**

(ii) $\Longleftrightarrow$ (i):

Clearly, $N$-reducibility is equivalent to $(1+A)^{N+1} = (1+A)^N$. From that we have:

$$(1+A)^{N+1} = (1+A)^N \iff (1+A)^N + A^{N+1} = (1+A)^N, \text{ using (3a)}$$

$$\iff A^{N+1} \leq (1+A)^N, \text{ using (3b)}$$

(i) $\Longrightarrow$ (iii):

Again, the starting point is $(1+A)^{N+1} = (1+A)^N$.

$$(1+A)^{N+1} = (1+A)^N \implies (1+A)^{N+2} = (1+A)^{N+1}$$

$$\implies (1+A)^{N+2} = (1+A)^N, \text{ by the } N\text{-reducibility of } A$$

Iterating, we get $(1+A)^{2N} = (1+A)^N(1+A)^N = (1+A)^N$.

(iii) $\Longrightarrow$ (ii):

We know that $(1+A)^{2N} = \sum_{k=0}^{2N} A^k$. Hence, (iii) implies that

$$\sum_{k=0}^{2N} A^k = \sum_{k=0}^{N} A^k \implies A^{N+1} \leq (1+A)^N, \text{ using (3b)}$$

From our previous discussion we have that the last formula implies the $N$-reducibility of $A$. □

**Proposition 3.2:** Any one of the following conditions is sufficient for $N$-reducibility (for some finite $N$):

(i) For some $m$, $A^m \geq A^m A^m$

(ii) For some $m$, $A^m \leq 1$

(iii) For some $m$ and for any initial relation $R$, $A^m R = \emptyset$ or $A^m R = CONST$, where $CONST$ some constant relation (i.e. $A^m R$ is in some sense independent of $R$).

**Proof:**

(i) Assuming the given condition we can derive the following:

$$A^m \geq A^m A^m \implies \sum_{k=0}^{2m} A^k = \sum_{k=0}^{2m-1} A^k, \text{ using (3b)}$$

$$\implies (1+A)(1+A)^{2m-1} = (1+A)^{2m-1}$$

$N$-reducibility is a direct consequence of this last formula.

(ii) If we multiply both sides of $A^m \leq 1$ by $A^m$ we get $A^{2m} \leq A^m$, which is the condition of (i) above. Applying that we conclude that $A$ is $N$-reducible.

(iii) If $A^m R = \emptyset$ then $A^m(A^m R) = \emptyset$ also. If $A^m R = CONST$ then $A^m(A^m R)$ can be equal to either $CONST$ or $\emptyset$ but in all cases it is true that $A^m(A^m R) \leq A^m R$ for all $R$. This implies $A^m \geq A^m(A^m)$ by definition. Hence we can apply condition (i) above and get $N$-reducibility. □

**Example:** As an example, consider the following Horn clause

$$salute(X,Y) \wedge captain(Y) \wedge major(Z) \rightarrow salute(X,Z),$$

i.e. "He who salutes a captain also salutes a major". The operator in this case can be expressed as

$$A\,S = \pi_1(S \bowtie C)\,x\,M$$

where $S$=*salute*, $C$=*captain*, $M$=*major*, $\pi_1$=project on the first column, $\bowtie$=join, $x$=cartesian product. We note that since the second column of $AS$ is $M$, we have

$$A^2 S = \begin{cases} A\,S & \text{if } M \cap C \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

Hence, $A^2 \leq A$ which implies $A^* = 1+A$, and $A$ is 1-reducible for this example. □

## 4. TRANSITIVE CLOSURE AS A PSEUDO-INVERSE

The lack of additive and multiplicative inverse impairs our ability to manipulate algebraic expressions of relational operators. In this regard, the transitive closure $A^*$ of a relational operator $A$ plays a useful role. We begin by noting that

$$A^* = \sum_{k=0}^{\infty} A^k = 1 + \sum_{k=1}^{\infty} A^k = 1 + AA^*$$

If we blindly solved for $A^*$, as if it was real number addition and multiplication in the equation,

we would get

$$(1-A)A^* = 1, \quad A^* = (1-A)^{-1} \tag{7}$$

However neither $1-B$ nor $B^{-1}$ is well defined for any relational operator $B$. Nonetheless, for any algebraic expression that can be simplified using $(1-A)^{-1}$, we can use $A^*$ in its place. For example, if we apply the above on the equation

$$C = AC + B$$

then we can get

$$C=A^*B$$

as a solution. For a more interesting example, consider the equation

$$A + CDA = B + CA + DA$$

Blindly solving for $A$, we get

$$(1 - C - D + CD)A = B$$

which can then be transformed by the appropriate factorization to

$$(1 - C)(1 - D)A = B$$

Multiplying both sides with the multiplicative inverses of $(1 - C)$ and $(1 - D)$ and applying (7) whenever possible yields

$$A = D^*C^*B$$

The fact that the above constitutes a solution to the original equation can be easily verified by a simple substitution of $D^*C^*B$ for $A$ in the equation. This solution for $A$ is hardly expected from the original equation.

The algebraic manipulation capability so afforded gives rise to the following important result.

**Theorem 4.1:** Let $A$ be such that its transitive closure satisfies the equation

$$A^* = B + CDA^*$$

Then,

$$A^* = [1 + C(DC)^*D]B$$

**Proof:** We multiply both sides of the equation by $D$.This yields

$$DA^* = DB + DC(DA^*)$$

If we now solve for $DA^*$ we get

$$DA^* = (DC)^*DB$$

Some additional manipulation allows the elimination of the front $D$ for the final solution:

$$A^* = B + C(DC)^*DB$$

□

**Corollary 4.1:** $(CD)^* = 1 + C(DC)^*D$

Query processing may be strongly affected by the result of Corollary 4.1, because the latter allows us to change the operator used in the iteration. In other words, if we can write some operator $A$ as a product of two terms, like $A = CD$, we can take the transitive closure of the symmetric operator $DC$ instead of $A^*$. Identifying cases where $DC$ is faster than $A$ will enable us to achieve faster processing time for the given query. For example, if $D$ produces a relation with fewer columns than $C$ then there is a good chance that $(DC)^*$ is faster than the original $(CD)^*$. Taking into account specific statistics about the database state at the given point of time, this swapping capability has the potential of being very useful to the query optimizer.

**Corollary 4.2:** $A^m = CD$ for some $m \Longrightarrow A^* = [1 + C(DC)^*D](1 + A)^{m-1}$

There seems to be a significant similarity between the results mentioned in this section and some results in regular expression equations. The relevant theorem, as mentioned and proved in [Denn78] ( also mentioned in [Chan81] ) says that the unique solution to the equation $A = B \bigcup CA$, with $B$ and $C$ regular expressions and $B$ not containing $\lambda$ (the empty string), is $A = C^*B$. This is exactly what Theorem 4.1 says for the same equation if we appropriately map union to addition, concatenation to multiplication and Kleene star to transitive closure. Whether a deeper relationship between relational operators and regular expressions exists or not remains to be seen.

As mentioned before, the result of Theorem 4.1 is particularly important for those cases where $D$ is a projection operator. In such cases $DC$ involves fewer variables than $A$ itself, and a decomposition is achieved. To identify such cases we now proceed to use a graph representation for recursive Horn clauses introduced in [Ioan85] .

## 5. GRAPH REPRESENTATION AND DECOMPOSITION

Consider a recursive Horn clause:

$$P(\underline{x}^{(0)}) \wedge Q_1(\underline{x}^{(1)}) \wedge \cdots \wedge Q_K(\underline{x}^{(K)}) \rightarrow P(\underline{x}^{(K+1)})$$

where each $\underline{x}^{(i)}$ is a subset of $\{x_1,x_2,...,x_n\}$. A graph representation (very similar to the one described in [Ioan85] ) can be constructed as follows:

(a) The nodes are in one-to-one correspondence with the variables $x_1,x_2,...,x_n$.

(b) A directed arc exists from each $x_i^{(0)}$ to $x_i^{(K+1)}$.

(c) An undirected arc exists from each $x_i^{(k)}$ to $x_j^{(k)}$, for all $1 \leq k \leq K$.

The arcs of the graph are weighted according to the following function:

weight(arc) = 0 if the arc is undirected

weight(arc) = 1 if the arc is directed and it is traversed in its direction

weight(arc) = -1 if the arc is directed and it is traversed in the opposite direction

For example, the Horn Clause

$$P(x,y) \wedge Q(y,z) \wedge R(x,z) \rightarrow P(y,z)$$

yields the graph of figure 1.

The weight of a path (cycle) in the graph is defined as the algebraic sum of the weights of the arcs along the path (cycle). Further, a cycle is said to be *weighted* if its weight is nonzero. For the remainder of this paper we make the assumption that

(A) "The directed arcs form a forest"

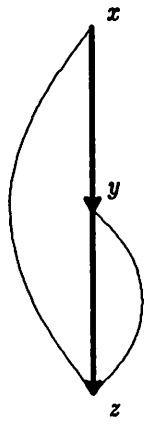


Fig. 1 : Example graph

Under this assumption, it was shown in [Ioan85] that the corresponding recursion is uniformly bounded if and only if the graph contains no weighted cycle.

We observe that the graph of a Horn clause can be transformed in the following ways without changing the underlying semantics.

(a) *Eliminate node* - Any node not involved in any directed arc can be eliminated. Regarding the logic formula, this is equivalent to performing a join or a semijoin and replacing the relations involved with the outcome after eliminating the join column. Notice that the join column is the one where the variable corresponding to the eliminated node appears. For example, $Q_1(x,z) \wedge Q_2(z,y)$ will be replaced with $Q_{12}(x,y)$ (see figure 2).


Fig. 2 : Eliminate node

(b) *Split node* - Any node can be replaced by two nodes connected by an undirected arc. This is equivalent to adding an "EQUAL" predicate in the Horn clause for the two variables corresponding to the two introduced nodes (see figure 3).

Fig. 3 : Split node

(c) *Combine directed arcs* - Two directed arcs can be combined into one as shown in figure 4.

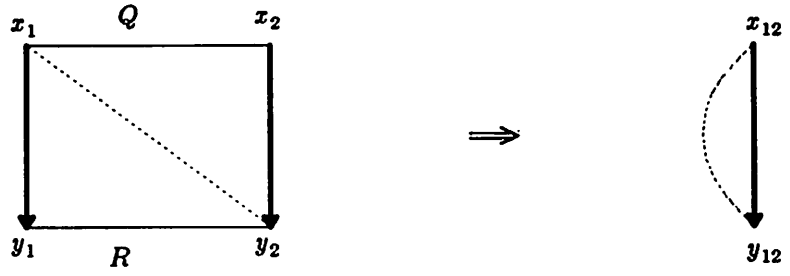


Fig. 4 : Combine directed arcs

Any undirected arcs that may connect the heads or the tails of the combined directed arcs get eliminated. This is equivalent to denoting $x_{12}=(x_1,x_2)$ and $y_{12}=(y_1,y_2)$.

We further note that (a) and (b) preserve all weighted cycles, whereas transformation (c) does not preserve weighted cycles in general.

Under assumption (A) a graph can be transformed while preserving weighted cycles into the following canonical form:

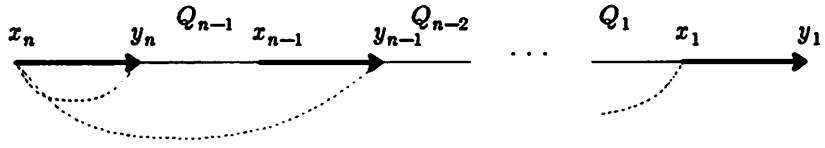


Fig. 5 : Canonical form of a graph

where the dotted lines indicate that additional undirected arcs may exist between any pair of nodes. However, for a graph corresponding to uniformly bounded recursion no such additional arc can exist since each one of them would imply a weighted cycle. We can write the Horn clause corresponding to a graph in canonical form as follows:

$$P(x_1,x_2,\ldots,x_n) \wedge Q_1(x_1,y_2) \wedge \cdots \wedge Q_{n-1}(x_{n-1},y_n) \wedge$$
$$R(x_1,x_2,\ldots,x_n,y_1,\ldots,y_n) \rightarrow P(y_1,y_2,\ldots,y_n)$$

For the uniformly bounded recursion case, $R$ takes the form $R = Q_0(y_1) \wedge Q_n(x_n)$ (see figure 6).

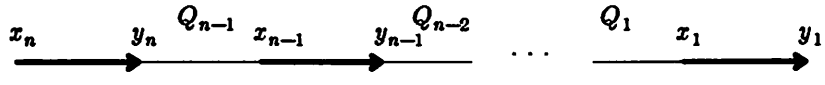


Fig. 6 : Canonical form of a graph for a bounded statement

Consider two special cases illustrated below:

(i)

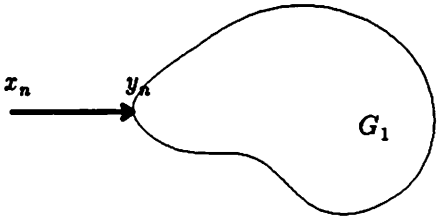


Fig. 7 : General form of decomposable graph with last column cycle-free

(ii)

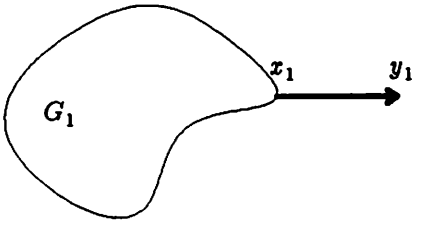


Fig. 8 : General form of decomposable graph with first column cycle-free

For case (i) $x_n$ does not appear as an argument in $R$ and the operator $A$ is of the form

$$A = \pi_n Q \ltimes \pi_{n-1}$$

where $Q$ is a combination of $Q_1, Q_2, \ldots, Q_{n-1}$ and $R$, $\ltimes$ is a left semijoin and $\pi_i$ denotes projection of the first $i$ columns. We can now apply Theorem 4.1 to find

$$A^* = 1 + \pi_n \, Q \ltimes (\pi_{n-1}\pi_n Q \ltimes)^* \pi_{n-1}$$

Projections $\pi_{n-1}$ and $\pi_n$ appear the one right after the other. This will give the same result as the last projection (that is, $\pi_{n-1}$) alone. Therefore, we can write the above as

$$A^* = 1 + \pi_n \, Q \ltimes (\pi_{n-1} Q \ltimes)^* \pi_{n-1}$$

Since $(\pi_{n-1} Q \ltimes)^*$ is a recursion involving $(n-1)$ variables, we have achieved a decomposition.

For case (ii), $R$ has the form

$$R'(x_1, \ldots, x_n, y_2, \ldots, y_n) \wedge S(y_1)$$

After one application of $A$ the first column of $P$ will forever remain $S$. In fact, after the first application of $A$, $P$ will be the cartesian product of $S$ with some other relation. Hence, $A^2$ can be

expressed as

$$A^2 = C\ \pi^{(n-1)}$$

where $\pi^{(n-1)}$ denotes projection on the last $(n-1)$ columns and $C$ uses relation $S$ directly instead of manipulating the first column of $P$. Applying Corollary 4.2 we can write

$$A^* = [1 + C\ (\pi^{(n-1)}\ C)^*\ \pi^{(n-1)}]\ (1 + A)$$

Once again, we have reduced an $n$-variable recursion to an $(n-1)$-variable recursion $(\pi^{(n-1)}\ C)^*$. In fact, further simplification is possible in this case. The operator $C$ does include some cartesian product with $S$ to produce an $n$-column relation. But this in completely useless in the inner loop $C$, since immediately afterwards the first column gets projected out. Hence, $C$ can be replaced by an even simpler $(n-1)$-column operator $D$ to finally produce for $A^*$ the formula

$$A^* = [1 + C\ D^*\ \pi^{(n-1)}]\ (1 + A)$$

Notice that the two cases above are only special cases of the more general decomposition mentioned above as a direct consequence of Corollary 4.1. Consider an arbitrary operator $A$. Assume that $A$ can be written as $A = C\,\pi D$, where $C$ and $D$ are some relational operators and $\pi$ is some projection producing fewer columns than $A$. From Corollary 4.1 we have that

$$A^* = (C\pi D)^* = 1 + C(\pi DC)^*\pi D$$

The result is that we have to take the transitive closure of an operator that produces relations with fewer columns than the original, making the whole operation potentially faster. In cases (i) and (ii) above, the acyclicity of some part of the corresponding graphs gave this projection. By breaking $A$ there, we produced more efficient transitive closure operators. Since any operator can be equivalently written in multiple ways using the basic operators (like join, project etc.), it is an interesting optimization problem to identify the best of these, in the sense of containing some projection on as few columns as possible. Of course, this cannot be the only consideration of the optimizer, since fewer columns doesn't necessarily mean faster execution. Nevertheless, it is a good indication of potentially fast access paths and we believe that it should be incorporated into a recursive query optimizer.

Finally, we shall obtain an explicit formula for $A^*$ when $A$ is $n$-reducible. We note that in such a case the canonical Horn Clause for $A$ is given by

$$P(x_1,x_2,\ldots,x_n) \wedge Q_0(y_0) \wedge Q_1(x_1,y_2) \wedge \cdots \wedge Q_{n-1}(x_{n-1},y_n) \wedge Q_n(x_n) \rightarrow P(y_1,y_2,\ldots,y_n)$$

Now, define the following operators:

$\pi_k =$ projection on first $k$ columns

$q_k = \pi_1(Q_k \ltimes Q_{k+1} \ltimes \cdots \ltimes Q_n)$

$C_k = Q_0 X Q_1 X \cdots X Q_k$

$R_Q =$ right semijoin with $Q$, i.e. $R_Q P = Q \rtimes P$

$L_Q =$ left semijoin with $Q$, i.e. $L_Q P = Q \ltimes P$

Then we can write

$$A = L_{C_{n-1}} \pi_{n-1} R_{q_n},$$

so that

$$A^* = 1 + L_{C_{n-1}}(\pi_{n-1} R_{q_n} L_{C_{n-1}})^* \pi_{n-1} R_{q_n}$$

However, $\pi_{n-1} R_{q_n} L_{C_{n-1}}$ can be written as

$$\pi_{n-1} R_{q_n} L_{C_{n-1}} = L_{C_{n-2}} \pi_{n-2} R_{q_{n-1}}$$

This equality holds for all indices from 2 to $n$, i.e.

$$\pi_{k-1} R_{q_k} L_{C_{k-1}} = L_{C_{k-2}} \pi_{k-2} R_{q_{k-1}}, \quad \forall\, 1 < k \leq n \tag{8}$$

Hence, if we define

$$A_k = L_{C_{k-1}} \pi_{k-1} R_{q_k}$$

then we have $A = A_n$ and by applying (8) continuously we get

$$A^* = A_n^* =$$

$$= 1 + L_{C_{n-1}} A_{n-1}^* \pi_{n-1} R_{q_n} =$$

$$= 1 + L_{C_{n-1}}(1 + L_{C_{n-2}} A_{n-2}^* \pi_{n-2} R_{q_{n-1}}) \pi_{n-1} R_{q_n}$$

.....

$$= 1 + L_{C_{n-1}}(\pi_{n-1}R_{q_n}) + L_{C_{n-1}}L_{C_{n-2}}(\pi_{n-2}R_{q_{n-1}})(\pi_{n-1}R_{q_n}) +$$

$$+ \cdots\cdots\cdots\cdots\cdots$$

$$+ L_{C_{n-1}}L_{C_{n-2}} \cdots L_{C_1}(\pi_1 R_{q_2}) \cdots (\pi_{n-2}R_{q_{n-1}})(\pi_{n-1}R_{q_n})$$

The above is an explicit expression of $A^*$ for any $n$-reducible operator $A$. Although the expression is complicated in the general case, it shows that a query involving bounded recursion can always be transformed into one that is recursion-free.

## 6. CONCLUSION

In general, recursion can be expected to represent a major source of processing inefficiency. Query optimization for recursion is ever more important than it is for non-recursive relational queries. In that connection two ideas are important: (a) precomputation and (b) decomposition.

The idea of precomputation is to augment the database with additional relations derived from the database so that some recursive queries on the original database are no longer recursive on the augmented one. The question is: for a given augmentation what is the set of such queries? And conversely, for a given class of recursive queries what must the augmentation be? Our results on this are fragmentary.

Decomposition is related to precomputation. The problem here is to reduce the complexity of recursion. For example, if $A$ can be expressed in terms of $B$ and $C$ both of which being simpler operators, when can $A^*$ be expressed in terms of $B^*$ and $C^*$?

Results pertaining to both precomputation and decomposition appear to be limited by the fact that relational operators form a semiring, rather than a ring. Perhaps, we need to embed the operators in a larger algebraic structure. In terms of processing, this may mean that more information will need to be kept at each stage of iteration (that is, assuming that recursive queries are answered by some kind of an iterative program), but the additional information requirements is compensated by greater applicability of precomputation and decomposition.

As a final comment we would like to emphasize the usefulness of the algebraic view taken here concerning recursive query optimization. We have already seen decomposition examples where simple algebraic manipulation produced equivalent, potentially more efficient operators. We believe that algebraic manipulation of the query will be a significant part of the query optimizer. This, along with the particular semantics of the individual query and statistics about the database at the given point of time will be used to produce the plan for the query execution. This is opposite from the case of the regular, non-recursive queries where, as experience has shown, algebraic manipulation of the query is of no help to the optimizer.

## 7. REFERENCES

[Chan81]
Chang, C. L., "On Evaluation of Queries Containing Derived Relations in a Relational Data Base", in *Advances in Data Base Theory* **Vol. 1**, edited by H. Galaire, J. Minker and J. M. Nicolas, Plenum Press, New York, N.Y., 1981, pages 235-260.

[Denn78]
Denning, P., J. Dennis, and J. Qualitz, *Machines, Languages and Computation,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1978.

[Gall78]
Gallaire, H. and J. Minker, *Logic and Data Bases,* Plenum Press, New York, N.Y., 1978.

[Ioan85]
Ioannidis, Y. E., "A Time Bound on the Materialization of Some Recursively Defined Views", *Proc. 11th International VLDB Conference,* Stockholm, Sweden, August 1985.