RANGE NESTING:

A FAST METHOD TO EVALUATE QUANTIFIED QUERIES (*)

Matthias Jarke
Graduate School of Business Administration
New York University

Jürgen Koch (**)
Fachbereich Informatik
Universität Hamburg
Schlüterstr. 70
D-2000 Hamburg 13
Federal Republic of Germany

December 1982

Center for Research on Information Systems
Computer Applications and Information Systems Area
Graduate School of Business Administration
New York University

Working Paper Series

CRIS #49

GBA #83-25(CR)

(*) Published in: Proceedings ACM-SIGMMOD International Conference on Management of Data, Database Week, San Jose/Ca., May 23-26, 1983.

(**) The work of Jürgen Koch was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant no. SCHM 450/2-1 (principal investigator: J.W. Schmidt).

Abstract

Database queries explicitly containing existential and universal quantification become increasingly important in a number of areas such as integrity checking, interaction of databases, and statistical databases. Using a concept of range nesting in relational calculus expressions, the paper describes evaluation algorithums and transformation methods for an important class of quantified relational calculus queries called perfect expressions. This class includes well-known classes of "easy" queries such as tree queries (with free and existentially quantified variables only), and complacent (disconnected) queries.

1. Introduction

Using explicit quantification in database queries has long been considered difficult to understand by users and inefficient to implement by the DBMS. However, several recent developments may lead to new interest in the neglected area of query optimization for quantified queries, especially for queries containing universal quantifiers.

First, there is a need to test general integrity constraints efficiently [BERN82]. Often, such constraints apply to all elements of a certain data set and therefore use universal quantification.

Second, more and more database systems are coupled with artificial intelligence systems. First-order predicate calculus with existential and universal quantifiers is one of the foundations of AI methods [NILS82]. It seems useful to provide similar tools in a database system to make the interaction efficient.

Third, very high-level user interfaces to database systems, such as natural language, make frequent use of quantification. A database programming language [SCHM82] that supports such interfaces as a target or implementation language should provide constructs to evaluate quantification efficiently.

Fourth, there is a growing interest to provide relational interfaces to DBMSs with other data models, e.g., networks. In such models, especially the notion of existential quantification seems very natural [DAYA82].

Finally, quantified queries are closely related to aggregate queries [KLUG82] playing an important role in the emerging area of statistical database management. In fact, most existing query evaluation systems implement quantification indirectly via aggregate functions (e.g., COUNT), if at all.

Our approach to query optimization for quantified queries makes direct use of the established body of first-order calculus research. The standard relational calculus [CODD72] is extended to allow the definition of so-called (range-) nested expressions.

Relational calculus expressions are transformed into nested expressions by rules that generalize the notion of extended range expressions introduced in [JARK82a]. Note that our concept of range nesting is different from the SQL concept of condition nesting [KIM82].

In extended range expressions, a range relation of the (tuple) relational calculus can be substituted by a monadic relation-valued expression over this relation. For example, an element variable can be bound to the subset of "professors" rather than to a full "employees" relation. This approach may lead to reduced costs for evaluating n-ary expressions by reducing the size of the participating relations.

Range nesting extends this idea by allowing the system to bind element variables to general relational expressions with one free relation variable but an arbitrary number of quantified variables. For example, a variable might be bound to "professors not teaching after 6pm", thus reducing further the size of the range relation and the complexity of the query structure.

Such a more specific definition of the scope of interest can be helpful for users in formulating queries and for the system in evaluating them. The specific goal of this paper is to define and optimize an important class of nested expressions, <u>perfect nested expressions</u> (PNE), and to characterize the corresponding class of relational calculus queries, <u>perfect expressions</u>. Examples of perfect expressions include tree queries [SHMU81] and complacent expressions [BERN82].

The paper starts with an overview of several representation forms useful to analyze and transform quantified queries. Section 3 introduces the concept of perfect nested expressions and analyzes algorithms for their efficient evaluation. Section 4 derives the possibilities to map relational calculus expressions into perfect nested expressions. Especially for queries containing universally quantified variables, ways to transform seemingly difficult queries into perfect expressions are developed.

2. Quantified Queries

Queries can be represented in a number of forms. A representation form suitable for the purpose of query optimization supports the analysis of the structural properties of the query and provides a well-defined basis for query transformation. A representation form also defines the class of queries under consideration.

In this section, three representation forms for quantified queries useful for different purposes are given: a relational calculus, a parse tree, and a so-called quant graph.

2.1 Relational Calculus Representation

A quantified query can be represented as a relation-valued (relational) expression [SCHM77]:

```
[EACH r1 IN R1,...,EACH rn IN Rn :
                <selection predicate>]
```

The ri are usually referred to as element variables and the Ri are called range relations.

The selection predicate is a first-order predicate over the variables of the target list and is completely defined by the following recursive rules:

1. Let ri.Ai denote the attribute Ai of variable ri, op ∈ [<,<=,>,>=,=,≠], and c a constant. Then

   (ri.Ai op c) is a monadic term, and
   (ri.Ai op rj.Aj) is a dyadic term.

2. Atomic predicates are defined as follows:

   (i) A term is an atomic predicate.
   (ii) TRUE is an atomic predicate.
   (iii) FALSE is an atomic predicate.

3. An atomic predicate is a selection predicate.

4. Let A be a selection predicate,
   r an element variable, and
   R a relation. Then

   (i) SOME r IN R (A)
   (ii) ALL r IN R (A)

   are selection predicates.
   A is said to define the scope of r.
   Terms of the form, Q r IN R, where

   $Q \in$ [EACH, SOME, ALL]

   are called quantified range terms.

5. Let A and B be selection predicates. Then

   (i) NOT (A) (negation)
   (ii) A AND B (conjunction)
   (iii) A OR B (disjunction)

   are selection predicates.

6. No other formulae are selection predicates.

The main advantage of the relational calculus representation is that it provides query analysis with access to the established body of predicate logic. Tables 2.1 and 2.2 contain the most relevant predicate calculus rules adapted to the many-sorted relational calculus.

Relational expressions are usually simplified and standardized in order to remove redundant subexpressions [HALL76] and to provide the evaluation procedure with a suitable starting point [CODD72], [PALE72], [WONG76]. A standard form that particularly supports the optimization and evaluation of independent subexpressions is the so-called disjunctive prenex normal form (DPNF).

Most transformations performed for the purpose of standardization and simplification are of a purely syntactic nature (see tables 2.1 and 2.2). However, the rules for quantifier movement (Q1 to Q4) required for the transformation into DPNF, and simplification based on empty relations (M7) are obviously data-dependent. Therefore, a compile-time approach to standardization and simplification as outlined in [JARK81] has to provide sufficient information so that quantifier movement can be corrected in the presence of empty relations. Simplification based on empty relations can not be performed at all at compile-time.

Alternatively, one can use a runtime algorithm as sketched below. It performs standardization and simplification just before query evaluation and, therefore, has more information about the actual data.

(1) Apply rules for empty relations (M7).

(2) Transform into DPNF
(Q1..Q4, Q9, Q1Ø, M3, M5, M6).

(3) FOR EACH conjunction DO

apply idempotency rules M4a to M4d.

(4) Apply idempotency rules M4e to M4i.

Assuming that A and B are selection predicates and that quantified expressions of the many-sorted relational calculus can be translated into equivalent (one-sorted) predicate calculus expressions according to

```
    SOME r IN R (A)                      {many-sorted}
      <==>
    SOME r ((r IN R) AND A)              {one-sorted}
and
    ALL r IN R (A)                       {many-sorted}
      <==>
    ALL r ((r IN R) ==> A)               {one-sorted}
```

then the following holds:

```
Q1: A AND SOME r IN R (B(r)) <==> SOME r IN R (A AND B(r))

Q2: A OR SOME r IN R (B(r))
      <==>
 a) SOME r IN R (A OR B(r))  | R ≠ []
 b) A                        | R = []

Q3: A AND ALL r IN R (B(r))
      <==>
 a) ALL r in R (A AND B(r))  | R ≠ []
 b) A                        | R = []

Q4: A OR ALL r IN R (B(r))   <==> ALL r IN R (A OR B(r))

Q5: SOME r1 IN R1 SOME r2 IN R2 (A(r1,r2))
      <==>
    SOME r2 IN R2 SOME r1 IN R1 (A(r1,r2))

Q6: ALL r1 IN R1 ALL r2 IN R2 (A(r1,r2))
      <==>
    ALL r2 IN R2 ALL r1 IN R1 (A(r1,r2))

Q7: SOME r IN R (A(r) OR B(r))
      <==>
    SOME r IN R (A(r)) OR SOME r IN R (B(r))

Q8: ALL r IN R (A(r) AND B(r))
      <==>
    ALL r IN R (A(r)) AND ALL r in R (B(r))

Q9: NOT ALL r IN R (A(r))   <==> SOME r IN R (NOT(A(r)))

Q10:NOT SOME r IN R (A(r))  <==> ALL r IN R (NOT(A(r)))

Q11:SOME r IN R (TRUE)      <==> TRUE, if R ≠ []

Q12:SOME r IN R (FALSE)     <==> FALSE

Q13:ALL r  IN R (TRUE)      <==> TRUE

Q14:ALL r  IN R (FALSE)     <==> FALSE, if R ≠ []
```

Table 2.1: Transformation rules for quantified expressions

```
M1: Commutative rules
 a) A OR B <==> B OR A      b) A AND B <==> B AND A
M2: Associative rules
 a) (A OR B) OR C   <==> A OR (B OR C)
 b) (A AND B) AND C <==> A AND (B AND C)
M3: Distributive rules
 a) A OR (B AND C)  <==> (A OR B) AND (A OR C)
 b) A AND (B OR C)  <==> (A AND B) OR (A AND C)
M4: Idempotency rules
 a) A AND A         <==> A
 b) A AND NOT(A)    <==> FALSE
 c) A AND TRUE      <==> A
 d) A AND FALSE     <==> FALSE
 e) A OR A          <==> A
 f) A OR NOT(A)     <==> TRUE
 g) A OR TRUE       <==> TRUE
 h) A OR FALSE      <==> A
 i) A OR (A AND B)  <==> A
M5: DeMorgan's rules
 a) NOT (A AND B)   <==> NOT(A) OR NOT(B)
 b) NOT (A OR B)    <==> NOT(A) AND NOT(B)
M6: Double negation rules
    NOT ( NOT(A) )  <==> A
M7: Empty relation rules
 a) [EACH r in []: A]        <==> []
 b) SOME r IN [] (A)         <==> FALSE
 c) ALL r IN [] (A)          <==> TRUE
```

Table 2.2: Transformation rules for general expressions

2.2 Parse Tree Representation

Graphical representation forms for queries have a number of attractive properties. The visual presentation of a query often leads to an easier understanding of its structural characteristics. In addition, graph theory offers a number of results useful for the analysis of graph structures (e.g., discovery of cycles, tree property, etc.).

A straightforward graphical representation of a general relational expression is its corresponding <u>parse tree</u>, in which quantified range terms, atomic predicates, and logical operators are represented as nodes, and syntactic relationships as edges. Example 2.1 illustrates the correspondence between a calculus expression and its parse tree.

<u>Example 2.1</u>: A relational calculus expression and its parse tree.

```
[EACH r IN R:
  ALL s IN S
    (pred1(r,s)
      AND
    SOME t IN T (pred2(s,t))
      AND
    SOME u IN U (pred3(s,u)))]
```

```
                     --------
                    |EACH r|
                    | IN R |
                     --------
                        |
                     --------
                    | ALL s|
                    | IN S |
                     --------
                        |
                      -----
                     |AND|
                      -----
              /         |         \
 ------------      --------      --------
|pred1(r,s)|      |SOME t|      |SOME u|
 ------------     | IN T |      | IN U |
                   --------      --------
                      |             |
               ------------   ------------
              |pred2(s,t)|   |pred3(s,u)|
               ------------   ------------
```

2.3 Quant Graph Representation

Special classes of queries, such as the class of conjunctive queries, play an important role in various approaches to query optimization [AHO79], [BERN81a], [CHAN77], [ROSE80]. In the following, we shall introduce the quant graph as a graphical representation for quantified conjunctive expressions.

A quant graph is a directed graph. Each node ni represents a quantified range term. According to the quantifier, we distinguish EACH-nodes, SOME-nodes and ALL-nodes. Each edge ni->nj represents a dyadic term. The direction of the edge indicates that the quantified range term nj is defined in the scope of ni.

A path between nodes ni and nj is a sequence of adjacent edges connecting both nodes. We distinguish undirected paths (denoted by ni--nj) in which the direction of the edges is irrelevant, and directed paths (denoted by ni-->nj) which may not contain two adjacent edges with opposite direction. A quant graph is connected, if for each node ni there is an undirected path ni--nj to every node nj ≠ ni in the graph.

A quant graph is called a tree, if there is a distinguished node r, the root, from which there is a unique directed path r-->ni to every node ni in the graph. A cycle is an undirected path connecting some node ni with itself. An absorber is a node having more than one incoming edges. According to the quantifier, we distinguish EACH-absorbers, SOME-absorbers, and ALL-absorbers.

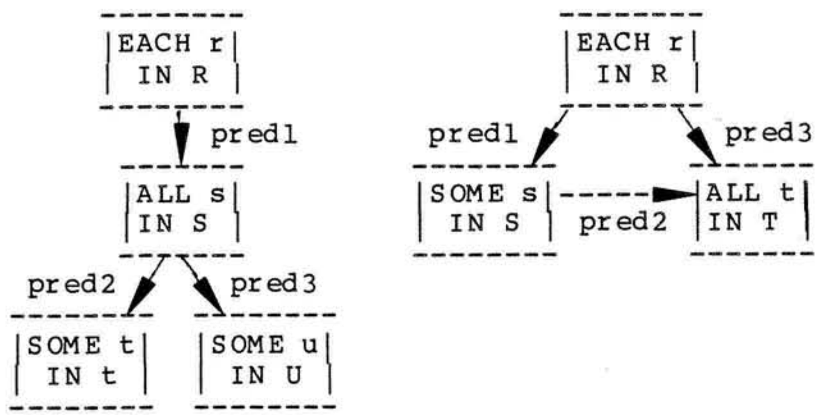
Lemma 2.1: A connected quant graph that is not a tree contains at least one absorber.

Proof: Assume that the graph does not contain an absorber. In this case, there is either a path from one node (the root) to all other nodes since all edges are directed, or the graph is disconnected. Both situations contradict the presupposition.

Lemma 2.2: Each cycle contains at least one absorber.

Proof: Follows directly from lemma 2.1.

Example 2.2: A quant tree and a cycle containing an ALL-absorber.



As defined here, quant graphs are more expressive than undirected graphs such as qual graphs [BERN81a], since they consider some of the scope rules of the predicate calculus. Extensions of the quant graph definition that cover all such rules are not needed in the context of this paper.

3. Perfect Nested Expressions

The representation forms introduced in section 2 are primarily useful for query analysis and query transformation. In addition, a representation form is needed, that relates relational expressions to a physical evaluation procedure. An extension to the relational calculus, the concept of nested expressions, can serve this purpose.

3.1 An Introductory Example

Before giving the general definition and evaluation procedure for nested expressions, we motivate our approach by a simple example.

Consider the relational database with the schema

```
EMPL (eno, ename, dno, status)
DEPT (dno, dname, city, street-address)
LECT (lno, eno, subject, time)
```

Suppose we are interested in "computer science departments employing professors who do not lecture after 6pm". This can be represented by the relational expression

```
[EACH d IN DEPT:
      d.dname = 'cs'
       AND
      SOME e IN EMPL
           (e.status = prof
             AND
            e.dno = d.dno
             AND
            ALL l IN LECT
               (l.time <= 6pm
                  OR
                 l.eno ≠ e.eno))]
```

The parse tree for this query is shown, below.

```
           ----------------
          |EACH d IN DEPT|
           ----------------
                  |
                -----
                |AND|
                -----
               /     \
              /       \
 --------------     ----------------
|d.dname='cs' |    |SOME e IN EMPL|
 --------------     ----------------
                           |
                         -----
                         |AND|
                         -----
                        /  |  \
   -----------------       |    ---------------
  |e.status = prof|        |   |e.dno = d.dno|
   -----------------       |    ---------------
                           |
                    ---------------
                    |ALL l IN LECT|
                    ---------------
                           |
                         -----
                         | OR|
                         -----
                        /     \
          -------------     ---------------
         |l.time<=6pm|     |l.eno ≠ e.eno|
          -------------     ---------------
```

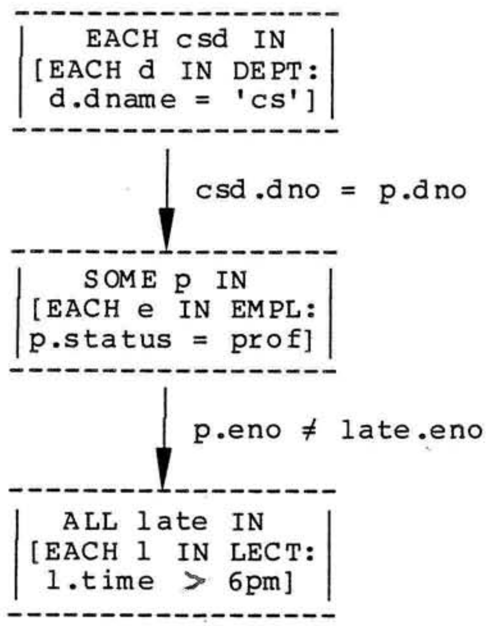
In [JARK82a], the concept of extended range expressions was introduced. Noting, that the query is interested only in

- computer science departments,
- professors, and
- lectures after 6pm,

the definition of the range relations can be extended to include the corresponding selective conditions and the query can be rephrased as

```
[EACH csd IN [EACH d IN DEPT: d.dname = 'cs']:
 SOME p IN [EACH e IN EMPL: e.status = prof]
  ALL late IN [EACH l IN LECT: l.time > 6pm]
   (csd.dno = p.dno AND p.eno ≠ late.eno)]
```

Below, a quant graph for this expression is shown.



The extension of this to range nesting can be motivated by addressing the question how a clever query optimizer would handle such a query. An efficient stepwise procedure might work as follows. The algorithm first finds out the late lectures:

[EACH l IN LECT: l.time > 6pm]

Next it would look for professors who do not teach late lectures. To do so, it embeds or "nests" the range definition of professors and late lectures into a more general expression describing professors who do not teach late lectures:

```
[EACH p IN [EACH e IN EMPL: e.status = prof]:
 ALL late IN [EACH l IN LECT: l.time > 6pm]
      (p.eno ≠ late.eno)]
```

So far, this is not different from the extended range expressions illustrated above. The next step of the procedure, however, looks for computer science departments employing the type of professor described above. This is expressed by nesting the definition of computer science departments <u>and</u> the complete expression describing "early lecture professors" into the more general expression whose value is the final answer to the original query:

```
[EACH csd IN [EACH d IN DEPT: d.dname = 'cs']:
 SOME earlylecturer IN
  [EACH p IN [EACH e IN EMPL: e.status = prof]:
   ALL late IN [EACH l IN LECT: l.time > 6pm]
      (late.eno ≠ p.eno)]
     (earlylecturer.dno = csd.dno)]
```

The example should make clear that the concept of nested range expressions yields an intermediate representation of a query which is equivalent to the original relational calculus query and defines important components of the query evaluation procedure. In the remainder of this section, this idea will be formalized and an important class of nested expressions will be defined which lends itself to particularly efficient evaluation.

3.2 Definition

A range-nested or, for short, nested expression is characterized syntactically by allowing to substitute the relation name, R, in a range term

```
[ ... r IN R ... ]
```

by a relational expression

```
[ ... r IN [ EACH r' IN .. : .. ] ... ].
```

Nested expressions are produced using the rules N1 to N3 that are motivated by the definition of the many-sorted calculus in table 2.1.

```
N1: [EACH r  IN R: pred1 AND pred2]
        <==>
    [EACH r  IN [EACH r' IN R: pred1]: pred2]

N2: SOME r  IN R  (pred1 AND pred2)
        <==>
    SOME r  IN [EACH r' IN R: pred1] (pred2)

N3: ALL r  IN R  (NOT(pred1) OR pred2)
        <==>
    ALL r  IN [EACH r' IN R: pred1] (pred2)
```

Expressions can be nested to arbitrary depth. The nesting generates a partial order on the evaluation of subexpressions, in that inner expressions are supposed to be evaluated before the outer ones.

Executing cheap selective operations first and serializing the execution of a query into a sequence of operations, each working on a single relation or variable, are heuristic approaches known to reduce the effort of query evaluation [SMIT75], [BERN81a]. We now define a class of nested expressions, namely perfect nested expressions, that are particularly well-suited for such heuristics.

A perfect nested expression (PNE) is defined recursively as follows:

(1) Let p(r) be a selection predicate with monadic terms only. Then

```
[EACH r IN R: p(r)]
```

is a PNE.

(2) Let A be a PNE and O(r) a one-level independent (OLI) selection predicate as defined below. Then

```
[ EACH r IN A: O(r) ]
```

is a PNE. An OLI is defined as follows. Let B be a PNE. Then

(2a)
```
SOME s IN B (p(r,s)) and
ALL  s IN B (p(r,s))
```

are OLI predicates where p(r,s) is a conjunction of dyadic terms.

(2b) If O1(r), O2(r) are OLI predicates, so are

```
(2b1) NOT ( O1(r) )
(2b2) O1(r) AND O2(r)
(2b3) O1(r) OR  O2(r).
```

To motivate this definition, consider some special cases. If only range relations of type (1) are used, the inner expressions are the extended range expressions of [JARK82a]. Substituting TRUE for p(r,s) in (2a) yields the complacent expressions of [BERN82].

If (1), (2a), and (2b2) are used, the expression is called a perfect conjunctive expression. The last version of the example in section 3.1 is a perfect conjunctive expression. The so-called tree queries [BERN81a], [SHMU81] are perfect conjunctive expressions with no universal quantifiers.

3.3. Evaluation

Perfect nested expressions can be evaluated in a bottom-up or in a top-down fashion following the nesting of expressions. This section describes the implementation of one step of the evaluation procedure. It should be noted, however, that additional efficiency can be gained by parallel processing of subexpressions refering to the same base relation [JARK82a]. Hardware-oriented approaches to the parallel execution of subexpressions are discussed in [VALD82].

Furthermore, it is often possible to pipeline successive steps of the evaluation, that is, to start the evaluation of an outer expression when only a partial result, e.g., one element, of the inner expression is available. In the sequel it is assumed that this is always done for a sequence of restrictions of the same relation. For example, the expression

[EACH r IN [EACH r' IN R: p(r')] : q(r)]

is evaluated like

[EACH r IN R: p(r) AND q(r)].

At each step of the procedure, a subexpression

[EACH r IN R: O(r)]

as defined in the previous subsection is processed. It is assumed that all the inner nestings in O(r) have already been evaluated. That is, O(r) consists of terms such as

SOME/ALL s IN VS (p(r,s))

where VS is the value set to which the inner expression computes on the given database state.

Consider first the case that p(r,s) is a single dyadic term (r.A op s.A). In this case, the transformations of table 3.1 can be applied to reduce the quantified expression to a single comparison. It can be seen that in all cases except J1 and J8 at most one value is returned by the inner expression. Therefore, a single monadic term is generated that restricts r.

In the cases J1 and J8, however, each element of R must be tested against the full value set. Considering the complete expression O(r), each r may have to be tested against a collection of value sets.

One way to do this in a bottom-up procedure, is to store all the value sets in hash tables or simply as ordered sequences allowing binary or, for larger value sets, tree search.

If there are only value sets of one attribute, an alternative to the above approach would be to conduct a merge join between the value sets and the outer relation. This requires sorting the relation by the given attribute first (unless the attribute is indexed) and may be worthwhile only if one of the value sets is very large.

```
Let A be an attribute of non-empty value set VS, VS[A] the projection
of the value set onto A, and s the variable of the outer relation.
Then the following transformations hold:

J1: SOME s IN VS (r.A = s.A)       ==> r.A IN VS[A]

J2: SOME s IN VS (r.A </ <= s.A)   ==> r.A </ <= MAX (VS[A])

J3: SOME s IN VS (r.A >/ >= s.A)   ==> r.A >/ >= MIN (VS[A])

J4: SOME s IN VS (r.A ≠ s.A)       ==> | TRUE, if |VS[A]| > 1
                                       | r.A ≠ VS[A], if |VS[A]| = 1

J5: ALL  s IN VS (r.A = s.A)       ==> | FALSE, if |VS[A]| > 1
                                       | r.A = VS[A], if |VS[A]| = 1

J6: ALL  s IN VS (r.A </ <= s.A)   ==> r.A </ <= MIN (VS[A])

J7: ALL  s IN VS (r.A >/ >= s.A)   ==> r.A >/ >= MAX (VS[A])

J8: ALL  s IN VS (r.A ≠ s.A)       ==> r.A NOT IN VS[A]
```

Table 3.1: Transformation rules for quantified join terms

More optimization is possible if some of the predicates in O(r) are AND-connected. Firstly, the predicates can be tested sequentially in decreasing order of selectivity to reduce the overall number of comparisons. Secondly, if indexes on the outer relation exist, the sets of element references generated by the value sets can be intersected before accessing the relation elements.

A top-down approach, as proposed by [KLUG82] for aggregate functions, uses the nested iteration method combined with the use of indexes. Obviously, this can be cost-effective only in the cases J1 and J8 since otherwise the generation of a single value may have to be repeated unnecessarily. Under the same assumption, the nested iteration method should also be used in a DBTG data structure as demonstrated for existentially quantified variables by [DAYA82]. There, the evaluation simply follows the set chains.

Now consider the case that p(r,s) is a conjunction of several join terms. If s is universally quantified, variable splitting (rule Q8) can be used to achieve the one term case. If s is existentially quantified, the terms must be evaluated simultaneously, and therefore a multi-attribute value set must be stored resulting in a more expensive search process. If nested iteration is used, feedback techniques as described in [CLAU80] may improve the efficiency of query evaluation.

In summary, the worst case time complexity of algorithms for perfect nested expressions is in the order of N log m for each step where m is the size of the value list and N is the size of the outer relation (at this step). As the overall bottom-up algorithm is sequential, this is also the overall complexity of the algorithm. Linear or near linear time is possible in all (one term predicate) cases where J1 and J8 do not occur, or if appropriate hashing functions for the value sets can be formed.

4. Transforming Quantified Queries Into Perfect Nested Expressions

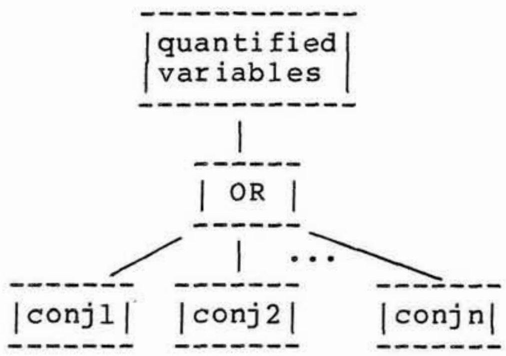
Certain classes of expressions, such as the example in section 3.1, are obviously equivalent to some perfect nested expressions and any clever query evaluation system would consider a stepwise evaluation procedure.

However, there are classes where this is not obvious at all and other classes, such as cyclic expressions, which look even rather hopeless [BERN81a]. A surprising result of our research is, that universal quantification, usually perceived as a trouble maker, often has a positive impact on expression nesting.

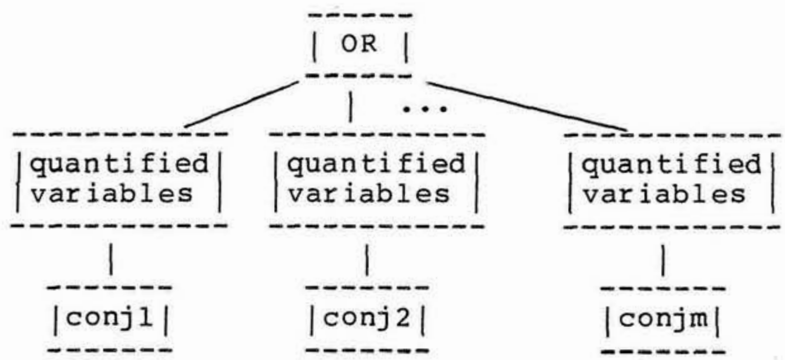
In this section, we investigate perfect expressions, defined as the class of expressions that can be transformed into perfect nested expressions. In particular, we are interested in the question how to nest a relational expression given in standard form. We shall deal with this problem in two steps. First, the essentials of separating an expression in standard form into (independent) conjunctive subexpressions are addressed. Second, some nesting properties of quantified conjunctive expressions are shown.

4.1 Separation

The parse tree of an expression in standard form is a chain of quantified variable nodes followed by an OR-node which branches out to the conjunctions of the matrix.



The desired form where the quantifiers directly precede the conjunctions would rather look like:



To achieve the desired form, the quantifiers must be distributed over the OR-nodes, one by one, starting with the innermost. Three types of transformation techniques can be used for this purpose:

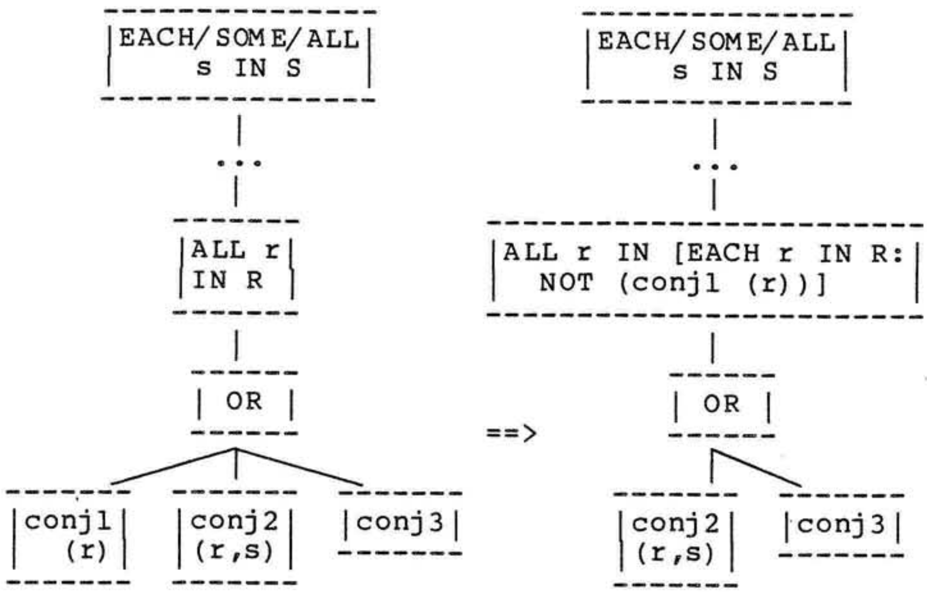
(1) <u>Variable Splitting</u> (according to rule Q7): This allows conjunctions over a common existentially quantified variable to be evaluated separately. The schema looks like

```
          ...                              ...
           |                                |
        --------                         ------
        |SOME r|                         | OR |
        | IN R |                         ------
        --------                         /    \
           |                      ---------   ---------
         ------                   |SOME r1|   |SOME r2|
         | OR |        ==>        | IN R  |   | IN R  |
         ------                   ---------   ---------
          / \                         |           |
 ----------  ----------        -----------  -----------
 |conj1(r)|  |conj2(r)|        |conj1(r1)|  |conj2(r2)|
 ----------  ----------        -----------  -----------
```

(2) Variable Propagation (according to rules Q2,Q4): This applies, if one or more conjunctions do not contain the variable r or variables defined in its scope.

```
           ...                           ...
            |                             |
       ----------                       ------
       |SOME/ALL|                       | OR |
       | r IN R |                       ------
       ----------                       /    \
            |                  ----------   -------
          ------               |SOME/ALL|   |conj2|
          | OR |      ==>      | r IN R |   -------
          ------               ----------
           / \                      |
 ----------  -------           ----------
 |conj1(r)|  |conj2|           |conj1(r)|
 ----------  -------           ----------
```

(3) Range Extension (according to rule N3): Technique (2) works only if there is just a single conjunction over r. Obviously, for existentially quantified variables this limitation can be removed by combining the first two rules. For universally quantified variables, the concept of range nesting can be applied in some cases to satisfy the precondition of (2). If a predecessor variable of r occurs in at most one conjunction together with r (e.g. in conjunction 2), a transformation as shown in the following schema yields a form that can be further transformed by means of (2):



If technique (3) is not applicable, the conjunctions must be evaluated simultaneously before evaluating the quantifier.

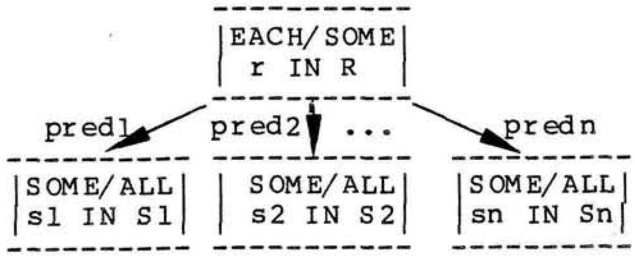
4.2 Amelioration

In this section we identify classes of conjunctive expressions, that can be transformed into equivalent perfect nested expressions, and that can thus be ameliorated with respect to evaluation performance. These classes will be characterized by means of structural properties of their corresponding quant graphs. Only expressions with connected quant graphs will be considered. Others (e.g., complacent expressions [BERN82]) can be evaluated piecewise. The results are generalizations of the well-known work on conjunctive expressions and tree expressions [CHAN77], [BERN81a], [ROSE8Ø] and are of particular interest because of the explicit exploitation of quantification.

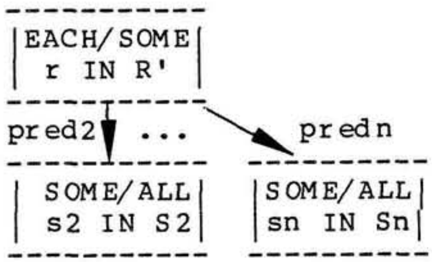
<u>Proposition 4.1</u>: Quantified conjunctive expressions whose quant graph is a tree are perfect.

<u>Proof Sketch</u>: The overall idea of the proof is to show how the expressions under consideration can be transformed into an equivalent perfect nested one. There are three types of subtrees distinguished by the quantification of the variable in the root node.

Types 1 and 2: Free and existentially quantified variables.



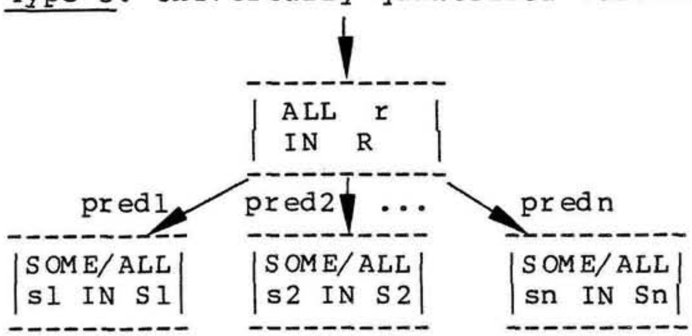
can be transformed to



where R' = [EACH r IN R:
SOME/ALL s1 IN S1(pred1(r,s1))]

The transformation follows rules N1 and N2. By induction on the breadth of the tree, it can be shown that trees of types 1 and 2 are perfect. If, at any time, R' becomes empty, the evaluation stops with a value of FALSE (SOME root) or an empty result (EACH root).

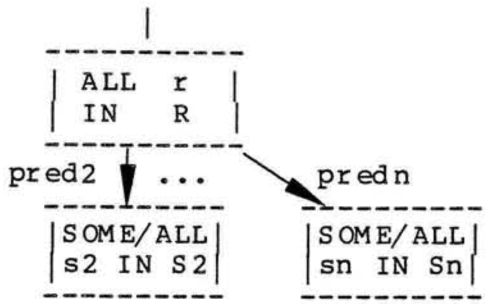
Type 3: Universally quantified variables.



For the processing of any edge, say the leftmost one, two cases must be distinguished.

```
case 1: [EACH r IN R:
          NOT (SOME/ALL s1 IN S1
               (pred1(r,s1)))]    =    []
```

In this case, the above subtree can be transformed according to rules Q8 and Q13:



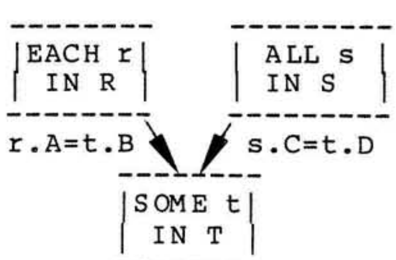

```
case 2: [EACH r IN R:
          NOT (SOME/ALL s1 IN S1
               (pred1(r,s1)))]    ≠    []
```

In that case, the evaluation procedure stops with a value of FALSE (rule Q14) for the subtree, and the entire tree evaluates to the empty relation.

By induction on the breadth of the tree it can be shown that trees of type 3 are perfect.

By induction on the height of the tree it can be shown that expressions whose quant graph is a tree are perfect.

If the quant graph is not a tree, it contains absorbers (lemma 2.1). The corresponding expressions are perfect only if there is a way to remove the absorbers. For example, the expression represented by the following graph is not perfect.

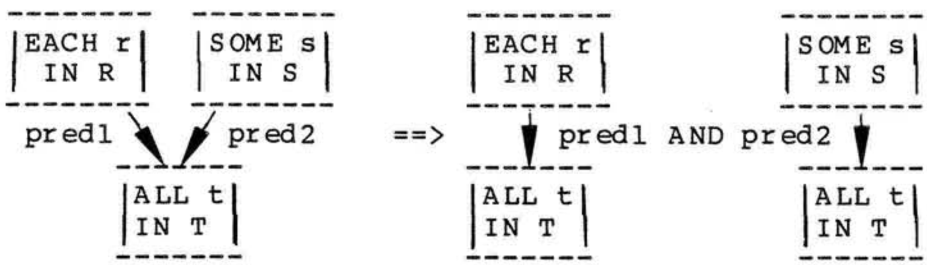


The SOME-absorber cannot be removed, since the quantifier sequence (ALL, SOME) must not be exchanged. In some cases however, absorbers can be removed by redirecting edges (rules Q1 to Q6) or by splitting absorber variables (Q7,Q8). For ALL-absorbers, the following lemma provides a powerful tool for query amelioration.

<u>Lemma 4.1</u>: ALL-absorbers can be removed by variable splitting.

<u>Proof</u>: Follows directly from rule Q8.

<u>Example 4.1</u>: Amelioration of a quant graph with a single ALL-absorber.



The expression of example 4.1 is decomposed into two disconnected subexpressions. If the right subexpression evaluates to FALSE, the overall result is the empty relation, otherwise it is the result of the left subexpression.

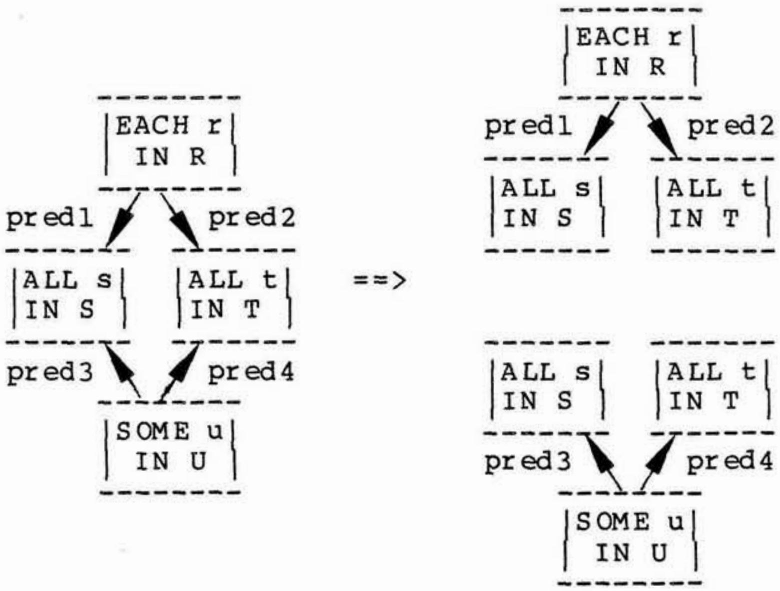
Cycles can occur in queries regardless of variable quantification. It is known that cycles containing EACH-absorbers and SOME-absorbers are not perfect in general [BERN81a]. However, some cycles can be broken to yield perfect expressions. Well-known examples are cycles induced by transitivity and cycles with certain inequality join terms [BERN81b].

For expressions with universally quantified variables, there is another way to break a cycle.

<u>Proposition 4.2</u>: An expression with a cyclic quant graph is perfect if the the cycle contains only ALL-absorbers.

<u>Proof Sketch</u>: Applying rule Q8, a cycle containing n ALL-absorbers is decomposed into n disconnected subtrees.

<u>Example 4.2</u>: Amelioration of a cycle with two ALL absorbers.



The class of perfect expressions is not limited to the cases discussed in this section. With certain combinations of universal and existential quantifiers, there are additional ways to remove SOME-absorbers. A detailed discussion would require an extension of the quant graph notation and is therefore beyond the scope of this paper, as is the analysis of non-conjunctive perfect expressions.

5. Conclusion

This paper introduced a concept of range nesting for the optimization of quantified queries, and outlined algorithms for the evaluation of perfect nested expressions. The class of perfect relational calculus expressions was shown to include the classes of quantified tree queries and of queries containing only ALL-absorbers (including cyclic queries).

Results for SOME-absorbers were mostly derived from earlier work. For space reasons, some more subtle details of the interaction between differently quantified variables are left to a forthcoming paper, as is the evaluation of non-perfect expressions. Another area of current research is the augmentation of the standardization and simplification phase to include semantic query optimization using integrity constraints.

The range nesting method is currently being implemented as a central logical query optimization strategy in the context of a calculus-based database programming language [JARK82b].

Acknowledgments

We are indebted to Manuel Mall for pointing out an error in an early draft of this paper. Thanks are also due to two of the referees whose detailed comments helped to improve the presentation.

References

AHO79 Aho, A.V., Sagiv, Y., Ullman, J.D. "Efficient Optimization of a Class of Relational Expressions", ACM-ToDS 4 (1979), 435-454.

BERN81a Bernstein, P.A., Chiu, D.M. "Using Semi-Joins to Solve Relational Queries", JACM 28 (1981), 25-40.

BERN81b Bernstein, P.A., Goodman, N. "The Power of Inequality Semijoins", Inform. Systems 6 (1981), 255-265.

BERN82 Bernstein, P.A., Blaustein, B.T. "Fast Methods for Testing Quantified Relational Calculus Assertions", Proc. ACM-SIGMOD Conf., Orlando, 1982, 39-50.

CHAN77 Chandra, A.K., Merlin, P.M. "Optimal Implementation of Conjunctive Queries in Relational Data Bases", Proc. 9th Annual ACM Symp. on Theory of Computation, Boulder, Col., 1977.

CLAU80 Clausen, S.E. "Optimizing the Evaluation of Calculus Expressions in a Relational Database System", Inform. Systems 5 (1980), 41-54.

CODD72 Codd, E,F. "Relational Completeness of Data Base Sublanguages", in Courant Computer Science Symposia, New York, Prentice Hall 1972.

DAYA82 Dayal, U., Goodman, N. "Query Optimization for CODASYL Database Systems", Proc. ACM-SIGMOD Conf., Orlando 1982, 138-150.

HALL76 Hall, P.A.V. "Optimization of Single Expressions in a Relational Data Base System", IBM J.Res. Develop. 20 (1976), 244-257.

JARK81 Jarke, M., Schmidt, J.W. "Evaluation of First-Order Relational Expressions", Technical Report 78, Universitaet Hamburg, Fachbereich Informatik, June 1981.

JARK82a Jarke, M., Schmidt, J.W. "Query Processing Strategies in the PASCAL/R Relational Database Management System", Proc. ACM-SIGMOD Conf., Orlando, 1982, 256-264.

JARK82b Jarke, M., Koch, J., Mall, M., Schmidt, J.W. "Query Optimization Research in the Database Programming Languages (DBPL) Project", Database Engineering 5, 3 (1982), 11-14.

KIM82 Kim, W. "On Optimizing an SQL-like Nested Query", ACM-ToDS 7 (1982), 443-469.

KLUG82 Klug, A. "Access Paths in the 'ABE' Statistical Query Facility", Proc. ACM-SIGMOD Conf., Orlando, 1982, 161-173.

NILS82 Nilsson, N. "Principles of Artificial Intelligence", Springer, 1982.

PALE72 Palermo, F.P. "A Data Base Search Problem", Proc 4th Comp. and Inform. Sc. Symp., Miami Beach, 1972, 67-1Ø1.

ROSE8Ø Rosenkrantz, D.J., Hunt, M.B. "Processing Conjunctive Predicates and Queries", Proc. 6th VLDB, Montreal, 198Ø, 64-74.

SCHM77 Schmidt, J.W. "Some High Level Language Constructs for Data of Type Relation", ACM-ToDS 2 (1977).

SCHM82 Schmidt, J.W., Mall, M., Koch, J., Jarke, M. "Database Programming Languages", Proc. Database Interface Workshop, Philadelphia, October 1982.

SHMU81 Shmueli, O. "The Fundamental Role of Tree Schemas in Relational Query Processing", PhD thesis, Harvard 1981.

SMIT75 Smith, J.M., Chang, P.Y.T. "Optimizing the Performance of a Relational Algebra Database Interface", CACM 18 (1975), 568-579.

VALD82 Valduriez, P. "Semi-Join Algorithms for Multi-Processor Systems", Proc. ACM-SIGMOD Conf., Orlando 1982, 225-233.

WONG76 Wong, E., Youssefi, K. "Decomposition - A Strategy for Query Processing", ACM-ToDS 1 (1976), 223-241.