AN EFFICIENT ALGORITHM FOR TESTING LOSSLESSNESS OF JOINS IN RELATIONAL DATA BASES

by

Lishing Liu
Alan Demers

TR78-351

Department of Computer Science
Cornell University
Ithaca, NY 14853

AN EFFICIENT ALGORITHM FOR TESTING LOSSLESSNESS OF JOINS IN RELATIONAL DATA BASES

by

Lishing Liu
Alan Demers

Cornell University

Abstract

Answering queries in a relational database model often requires the computation of joins of relations. Losslessness of joins is an important property for joins of relations to be semantically meaningful. In this paper we present an $O(n^3)$ algorithm for testing losslessness of joins in relational databases with functional dependencies, which improves the $O(n^4)$ result by Aho, Beeri and Ullman.

1. Introduction

Answering queries in a relational database often requires the computation of "joins" of relations. However, not all joins are semantially meaningful. In [ABU] a formal model of relational databses is given, and an $O(n^4)$ algorithm to test for "losslessness" of joins is outlined. Readers are encouraged to refer to [ABU] for formal definitions of relations, joins and functional dependencies in relational databases. For our purpose we need only describe a modified version of the losslessness testing algorithm introduced in [ABU]. Informally, the problem is as follows:

Let $A = \{1,2,\ldots,k\}$ be a set of "attributes' in a relational database model. Let R1, R2, ... , Rm be m "relation schemes", each of which is a non-empty subset of A, and let $R = \bigcup_i Ri$. Let r1, r2, ... , rl be l "functional dependency rules", each of which is of the form X --> Y, where X and Y are disjoint non-empty subsets of A. For each rule r: X-->Y we denote X (Y resp.) as lhs(r) (rhs(r) resp.). For given attributes $A = \{1,2,\ldots,k\}$ and relation schemes {R1,...,Rm} we initialize an mxk matrix T of symbols such that

$$T[i,j] = aj \quad \text{if } j \text{ is in } Ri,$$
$$= bij \quad \text{otherwise.}$$

We want to equate symbols in T using given dependency rules. Consider the procedure apply:

```
procedure  apply(r_s,i,i')
  begin
    if  T[i,j]=T[i',j] for all j in lhs(r_s)  then
       for  q in rhs(r_s)  do
          equate T[i,q] and T[i',q]
          rof
       fi
    end  apply
```

In apply, if two rows i and i' of T have equal symbols on each of the columns of $lhs(r_s)$ then we can equate the symbols of these rows in each column of $rhs(r_s)$.

Let us consider the problem TL:

"For given attributes, dependency rules and relation schemes, can all the entries on columns indicated by R of a certain row of T be equated to a_j's by successive calls to the procedure apply ?"

IN [ABU] it is shown that testing losslessness of joins in relational databases with functional dependencies is equivalent to solving the problem TL.

2. An Efficient Algorithm for TL

With some thought the reader should be able to convince himself that the equating process described in Section 1 has the Church-Rosser property [Se]. That is, if two sequences of equating steps are applied to a certain given table, the two resulting tables can always be equated into the same table (up to renaming of symbols) from successive calls of apply. Furthermore, successive calls of apply can result in a limiting table in which no more symbols can be equated. By the Church-Rosser property, this limiting table is unique up to renaming of symbols.

In order to solve TL efficiently we define some arrays and linked lists below. Recall that m, k and l are the number of relation schemes, the number of attributes, and the number of dependency rules respectively.

cnt - an mxmxl array of integers.
$cnt[i,i',s] = \#\{j \in lhs(r_s) \mid T[i,j] \neq T[i',j]\}$,
where # is the cardinality operator.

llsit - an array such that, for $1 \leq j \leq k$, llist[j] contains a list of all rules r_s with $j \in lhs(r_s)$.

In this section we will give an algorithm, TEST, which performs the symbol equating and solves the problem TL. TEST runs in a manner similar to the topological sorting algorithm [Kn]. A queue Q is maintained to hold triples (i,i',j) such that T[i,j] and T[i',j] are to be equated. In the following, for symbols d and e in table T, d<>e denotes that d and e have been equated. For $1 \leq i \leq m$ and $1 \leq j \leq k$, C([i,j]) denotes the set {i' | T[i',j]<>T[i,j], $1 \leq i' \leq m$}.

The initialization part of TEST is as follows:

```
for  1≤i<i'≤m, 1≤s≤l  do        /* initialize  cnt */
    cnt[i,i',s] := #lhs(r_s)
    rof

for  1≤j≤k  do                  /* initialize  llist's */
    llist[j] := null_list
    rof
for  1≤s≤l  do
    for  j∈lhs[r_s]  do
        push  s  to llist[j]
        rof
    rof

for  1≤i≤m, 1≤j≤k  do           /* initialize  T */
    if  j∈Ri  then  T[i,j] := aj
              else  T[i,j] := bij
        fi
    rof

Q := empty_queue                 /* initialize  Q */
for  1≤j≤k  do
    for  1≤i<i'≤m with T[i,j]=T[i',j]=aj  do
        push  [i,i',j]  on Q
```

```
        rof
    rof
```

Notice that, when initializing cnt, we only initialize cnt[i,i',j] for i<i'. The reason is that, according to the definition of cnt, cnt[i,i',j]=cnt[i',i,j] and cnt[i,i,j]=0. There is no need to keep duplicated values.

The main loop of TEST is as follows:

```
while  Q not empty  do/* more symbols to be equated? */
   pop an element [i,i',j] off Q
   if not(T[i,j]<>T[i',j])  then
      for h∈C(T[i,j]), h'∈C(T[i',j]) and s∈llist[j]  do
         if h<h' then cnt[h,h',s] := cnt[h,h',s]-1
                 else cnt[h',h,s] := cnt[h',h,s]-1
            fi
         /* Performs apply(r_s,h,h') */
         if cnt[h,h',s] or cnt[h',h,s] reaches 0 then
            for  j'∈rhs[r_s]  do
               push [h,h',j] on  Q
               rof
            fi
         rof
      equate T[i,j] and T[i',j]
      fi
      end  while
```

The last part of the algorithm TEST is

12 <u>if</u> for some i, $1 \le i \le m$, T[i,j]<>aj for all j in R
<u>then</u> print("Yes")
<u>else</u> Print("No")

Let us consider the main loop of TEST. Statements 7 - 10 essentially perform the procedure call apply(r_s,h,h'). Instead of equating immediately T[h,j] and T[h',j] as in the procedure body of apply, statement 9 pushes [h,h',j] on Q so that the side-effect of equating these two symbols will be handled properly when [h,h',j] is popped off from Q in the while-loop.

When statmet 4 is reached T[i,j] and T[i,j'] are to be equated at statement 11. Before equating these two symbols the array cnt is updated, and the procedure apply is called to handle possible new instances of symbols to be equated.

<u>Theorem 1</u>:

For given attributes, relation schemes and depndency rules, algorithm TEST prints "Yes" if TL has a positive answer, and it prints "No" if TL has a negative answer.

<u>Proof</u>: First we show that the main loop always terminates. In each iteration of the loop an element is always popped off from Q at statement 2. Now New elements are pushed on Q at statement 9 only when an entry of cnt is just decremented to 0. But each entry of cnt can be decremented to 0 at most once. Hence no new elements can be

pushed on Q eventually and the main loop will terminate afterwards.

From the structure of the main loop it is easily seen that symbols are equated according to the rule of apply and that the array cnt is always updated correctly. When two symbols T[i,j] and T[i',j] are equated aat statement 11 all the possible new instances of symbol equating are handled properly at statement 9. Therefore, when the main loop terminates with Q empty, no more symbols in T can be equated by calling apply. From our argument in Section 1 we see that the output of TEST correctly interpretes the answer to problem TL. □

To caoulate te running time of TEST we look at the total possible time spent on the execution of each statement of TEST.

<u>Theorem</u> <u>2</u>:

Algorithm TEST terminates in $O(n^3)$ steps, where n is the space needed to write down the input attributes, relation schemes and dependency rules.

<u>Proof</u>: The initialization part of TEST can obviously be done in $O(n^3)$ elementary steps. We only need to consider the execution time required by the main loop.

Whenever the test at statement 3 is true two symbols are always equated at statement 11. Now since "<>" is an equivalence relation and there are $O(n^2)$ entries in T, the test at statement 3 can yield a true condition at most $O(n^2)$ times. Therefore statements 4 and 11 can be executed at most $O(n^2)$ times.

Now consider statements 5 - 10. For each fixed j, the situation when statement 5 is reached for a particular pair(h,h') is that, for some i, T[i,j] and T[i',j] are not equated yet, T[h,j]<>T[i,j], and T[h',j]<>T[i',j]. But T[i,j] and T[i',j] are always equated aat statement 11 right after the for-loop beginning at statement 5. Hence, for a particular j and a particular pair (h,h') statement 5 can be reached at most once. Therefore, statement 5 can be executed at most $m^2 * \sum_{1\leq j\leq k}$ (#llist[j]) times. Now $\sum_{1\leq j\leq k}$ (#llist[j]) $\leq \sum_{1\leq s\leq l}$ (#lhs[r_s]), since the description of each rule r_s in the input stream contains a list of all attributes in lhs[r_s]. Therefore (#llist[j]) is O(n), and hence the total running time spent on statements 5 - 8 is $O(n^3)$.

It is left to consider statement 9. Let us consider the situation when the if-condition at statement 7 is true. Without loss of generality we may assume that h < h'. Now, for particular h, h' and s, cnt[h,h',s] can be decremented to 0 at most once. Hence statment 9 can be executed at most $m^2 *$ (#rhs(r_s)) times, which is clearly $O(n^3)$.

Therefore the total running time of TEST is $O(n^3)$. □

3. Conclusion

In the previous section we have shown that, for given attributes, relation schemes and functional dependency rules, the problem TL can be solved in $O(n^3)$ time. From our discussion in Section 1 we can conclude that the test of losslessness of joins in relational databases with functional dependencies can be done in $O(n^3)$ time. Our result improves the $O(n^4)$ result announced in [ABU].

References

[ABU] Aho, A.V., Beeri, C. and Ullamn, J.D., "The Theory of Joins in Relational Data Bases", _Proc. 18th Annual IEEE Symposium on Foundations of Computer Science_, Providence, Rhode Island, Oct. 1977, pp. 107-113.

[Kn] Knuth, D.E., _The Art of Computer Programming_, _Vol. I: Fundamental Algorithms_, Addison-Wesley, Reading, Massachusetts [1968].

[Se] Sethi, R., "Testing for Church-Rosser Property", _JACM_, 21:4, October 1974.