Annales Univ. Sci. Budapest., Sect. Comp. 15 (1995) 125-141

AN EFFICIENT SEMI–NAIVE ALGORITHM FOR DATALOG

T. Márkus and Manh Thanh Le
(Budapest, Hungary)

Abstract. This paper gives the Efficient Semi-Naive Algorithm for the Datalog program which is a modification of the Semi-Naive Algorithm. This modified algorithm converges faster than the semi-naive one and has no redundancies of the join operation. At the same time the paper also introduces a stratified method for Datalog programs. By the method we can use several variants of the naive and semi-naive algorithms to evaluate Datalog programs.

1. Introduction

The use of the logic programming paradigm in the context of data and knowledge base has been an important focus of research in the last years. This research revolved around attempts to extend Datalog.

We can view a Datalog program as defining functions that take EBD relations as arguments and IDB relations as values. Likewise, relational algebraic expressions defining functions take given relations as arguments and produce values which are the computed relations.

In Chapter 3 of [1] it is noted that the set of functions expressible in relational algebra is equivalent to those functions which are expressible in the Datalog (with possible negation), if the rules are rectified to be safe, non-recursive.

One of main characteristics of logic programming languages is their ability to express recursion. In this case the fixed point theory is well suited for describing their semantics ([2]).

Research was supported by the Hungarian Foundation for Scientific Research Grant OTKA 2149.

For each rule of a (possibly recursive) Datalog program we can correspond a relational algebraic equation. The meaning of a Datalog program is the least fixed point of the corresponding system of relational algebraic equations. This paper is dealing with the efficient solution of the system of equations.

Datalog programs are built from atomic formulae: ordinary and built-in predicates.

The ordinary predicate is a predicate symbol with a list of arguments which can be constants and variables. According to one predicate p there is a relation P such that a tuple $(a_1, a_2, \ldots, a_n) \in P$ iff $p(a_1, a_2, \ldots, a_n)$ is true. If the relation P is given in the database, then the corresponding p is an EDB (Extensional DataBase) predicate, the other ordinary predicate is an IDB (Intensional DataBase) one.

The built-in predicate is an arithmetic comparision.

An atomic formula or its negation is a literal. If all arguments in the literal are constants, then it is a ground literal.

A Datalog program is a collection of Horn-clauses in the form of $q : -s_1 \& s_2 \& \ldots \& s_n$, where q must be an IDB predicate (head of the rule), s_i's $(\overline{1,n})$ are positive literals, which are subgoals of the body $s_1 \& s_2 \& \ldots \& s_n$. The meaning of this rule is: if $s_1 \wedge s_2 \wedge \ldots \wedge s_n$ is true, then q must be true, here $\wedge$ means the operation of conjunction.

For showing dependencies of predicates in a program we draw a dependency graph whose nodes are the ordinary predicates and there is an arc from predicate p to predicate q if there is a rule with the head q and a subgoal p. A Datalog program is recursive if its dependency has one or more cycles (loops).

We say that predicates p and q are mutually recursive in the program **P** if they are both contained in a circle of the dependency graph of **P**.

To each rule $r : q : -s_1 \& s_2 \& \ldots \& s_n$ there corresponds a relational algebraic equation $Q = \mathrm{EVAL} - \mathrm{RULE}(r, P_1, \ldots, P_\beta)$, where P_i's $(i = \overline{1,\beta})$ are the relations of the ordinary predicates s_i's occuring in the body of the rule r, and EVAL-RULE(.) is a relational algebraic expression containing operations: join, projection, selection. This expression is introduced in [1].

For the set of rules having the same head predicate $r_k : q : -s_{k1} \& s_{k2} \& \ldots \ldots \& s_{kn_k}$ $(k = \overline{1,m})$ the corresponding equation is $P_i = \mathrm{EVAL}(q, P_1, \ldots, P_\beta)$, where $\mathrm{EVAL}(q, P_1, \ldots, P_\beta)$ is the union of EVAL-RULE$(r_k, .)$ $(k = \overline{1,m})$ after taking appropriate projections. For convenience we sometimes write $E_i(P_1, \ldots, P_\beta)$ instead of $\mathrm{EVAL}(p_i, P_1, \ldots, P_\beta)$.

We give some definitions for classification of a Datalog program and the system of corresponding relational algebraic equations.

Definition 1.1. Given a Datalog rule $r : q : -s_1 \& \ldots \& s_n$. We say that r is linear with respect to s_i if there is at most one occurence of s_i in the body of the rule.

Definition 1.2. Given a Datalog program **P** containing the rule $r : q : -s_1 \& s_2 \& \ldots \& s_n$. We say that r is linear in the program **P** if there is at most one s_i in the rule body (possibly q itself) which is mutually recursive to q in **P**.

Definition 1.3. A Datalog program **P** is linear if

i) all its rules are linear in **P**;

ii) the dependency graph of **P** contains at most one circle (loop).

Definition 1.4. An equation $P_i = E_i(P_1, \ldots, P_\beta)$ is linear with respect to P_j iff it is derived from the set of Datalog rules which are linear in the predicate p_j corresponding to P_j.

Definition 1.5. Given a Datalog program **P** with the IDB predicates $p_1, p_2, \ldots, p_\beta$ in it, then the corresponding system of equations $P_i = E_i(P_1, \ldots, P_\beta)$ $(i = \overline{1, \beta})$ is linear iff the program **P** is linear.

2. Buttom-up evaluation methods for finding the least fixed point (LFP) of a Datalog program

2.1. The Naive evaluation

The well known and simplest method for finding LFP is the following

Algorithm 2.1. *Naive algorithm* ([1])

Input: The system of relational algebraic equations $P_i = E_i(P_1, P_2, \ldots, P_n)$ $(i = \overline{1, n})$ of a Datalog program **P** and an Extensional Database $EDB = (R_1, R_2, \ldots, R_m)$.

Output: The Intensional Database $IDB = (P_1, P_2, \ldots, P_n)$ as least fixed point of **P**.

Method: The details of the algorithm are shown in Figure 2.1.

```
for i := 1 to n do P_i = ∅;
    repeat
    for i := 1 to n do B_i := P_i;
    for i := 1 to n do
```

$P_i := E_i(B_1, B_2, \ldots, B_n)$;
until $P_i = B_i$ for all i;
output $P_i's$

Figure 2.1

This method has a number of redundancies of the join operations in computing $P_i^{(k)}$, where $P_i^{(k)}$ is the result of relation P_i after k executions of repeat-loop. It is easy to see that Algorithm 2.1 halts and its correctness is stated by the following

Theorem 2.1. ([1]) *Algorithm 2.1 correctly computes the least model of a rectified safe Datalog program.*

2.2. The Semi-Naive evaluation

To decrease the number of redundancies of the join operations in the computation the well known Semi-Naive algorithm is used. In this algorithm the incremental relations are used to avoid computing again the tuples computed already.

First we consider a rectified safe Datalog rule r of the form

$$(1) \qquad r: \quad p_i(X_1, X_2, \ldots, X_k) : -s_1 \& s_2 \& \ldots \& s_m \& s_0,$$

where s_i's ($i = \overline{1,m}$) are ordinary subgoals; s_0 is a list of built-in predicates in the rule.

For each i ($i = \overline{1,m}$) let R_i be the current relation of the literal s_i, ΔR_i be the set of tuples in R_i on the most recent round of the Naive algorithm. For each rule r we define the incremental relation as follows

$$\text{EVAL} - \text{RULE} - \text{INCR}(r, R_1, R_2, \ldots, R_m, \Delta R_1, \Delta R_2, \ldots, \Delta R_m) =$$
$$= \bigcup_{1 \le i \le m} \text{EVAL} - \text{RULE}(r, R_1, R_2, \ldots, R_{i-1}, \Delta R_i, R_{i+1}, \ldots, R_m).$$

Now suppose that P_i's ($i = \overline{1,n}$) are relations for IDB predicates and ΔP_i is the incremental relation associated to P_i. We denote the set of what EVAL-RULE-INCR produces for all rules with head p_i, after taking a projection onto variables $X_1, X_2, \ldots, X_k$ by

$$\text{EVAL} - \text{INCR}(p_i, P_1, P_2, \ldots, P_n, \Delta P_1, \Delta P_2, \ldots, \Delta P_n).$$

Algorithm 2.2. *Semi-Naive algorithm* ([1])

Input: The system of relational algebraic equations $P_i = E_i(P_1, P_2, \ldots, P_n)$ $(i = \overline{1,n})$ of a Datalog program **P** and an Extensional Database EDB= $= (R_1, R_2, \ldots, R_m)$.

Output: The Intensional Database IDB=$(P_1, P_2, \ldots, P_n)$ as least fixed point of **P**.

Method: The details of the algorithm are shown in Figure 2.2.

for $i := 1$ to n do begin $P_i := \emptyset$; $\Delta P_i := E_i(\emptyset, \emptyset, \ldots, \emptyset)$ end;
repeat
  for $i = 1$ to n do $\Delta B_i := \Delta P_i$;
  for $i = 1$ to n do
    begin
    $\Delta P_i := \text{EVAL} - \text{INCR}(p_i, P_1, P_2, \ldots, P_n, \Delta B_1, \Delta B_2, \ldots, \Delta B_n)$;
    $\Delta P_i := \Delta P_i - P_i$
    end;
  for $i = 1$ to n do $P_i := P_i \cup \Delta P_i$;
until $\Delta P_i = \emptyset$ for all i

Figure 2.2

Theorem 2.2 ([1]) *The algorithm 2.2 correctly computes the least model of a rectified safe Datalog program.*

2.3. Evaluation of Datalog programs in special forms

In that case when each equation of the system is linear with respect to every variable P_i $(i = \overline{1,n})$, the equation is derived from rules which are linear with respect to all predicates in rule bodies. We consider a rule r of form (1) and suppose that each IDB predicate has no more one occurence in the body. We have

$$\text{EVAL} - \text{INCR}(p_i, P_1, P_2, \ldots, P_n, \Delta P_1, \Delta P_2, \ldots, \Delta P_n) =$$

$$\bigcup_{r \in H_r} \Pi_{X_1 \ldots X_k} \left[\bigcup_{1 \le i \le n} \text{EVAL} - \text{RULE}(p_i, P_1, \ldots, P_{i-1}, \Delta P_i, P_{i+1}, \ldots, P_n) \right] =$$

$$\bigcup_{1 \le i \le n} \Pi_{X_1 \ldots X_k} \left[\bigcup_{r \in H_r} \text{EVAL} - \text{RULE}(p_i, P_1, \ldots, P_{i-1}, \Delta P_i, P_{i+1}, \ldots, P_n) \right] =$$

$$\bigcup_{1 \le i \le n} E_i(P_1, \ldots, P_{i-1}, \Delta P_i, P_{i+1}, \ldots, P_n),$$

where H_i is the set of rules having the head predicate p_i in the program.

Thus for the Semi-Naive algorithm we can use the following iterations

$$P_i^{(k+1)} = P_i^{(k)} \cup \Delta P_i^{(k+1)},$$

where

$$\Delta P_i^{(k+1)} = \bigcup_{1 \le i \le n} E_i(P_1^{(k)}, \ldots, P_{i-1}^{(k)}, \Delta P_i^{(k)}, P_{i+1}^{(k)}, \ldots, P_n^{(k)}) - P_i^{(k)},$$

$$\Delta P_i^{(0)} = P_i^{(0)} = \emptyset \qquad (i = \overline{1,n};\ \ k = 0,1,2,\ldots).$$

In that case if each equation $P_i = E_i(P_1, P_2, \ldots, P_n)$ is linear with respect to P_i itself, then instead of the iterations of Semi-Naive algorithm we can use the following simpler iterations:

$$P_i^{(k+1)} = P_i^{(k)} \cup \Delta P_i^{(k+1)},$$

where

$$\Delta P_i^{(k+1)} = E_i(P_1^{(k)}, P_2^{(k)}, \ldots, P_{i-1}^{(k)}, \Delta P_i^{(k)}, P_{i+1}^{(k)}, \ldots, P_n^{(k)}) - P_i^{(k)},$$

$$\Delta P_i^{(0)} = P_i^{(0)} = \emptyset \qquad (i = \overline{1,n};\ \ k = 0,1,2,\ldots).$$

Example 2.1. Given the Datalog program **P**

$$p_1(X,Y) : -s_1(X,Y).$$
$$p_1(X,Y) : -s_1(X,Z)\&p_1(Z,Y).$$
$$p_2(X,Y) : -s_2(X,Y).$$
$$p_2(X,Y) : -s_2(X,Z)\&p_2(Z,Y).$$
$$p_3(X,Y) : -p_1(X,Z)\&p_2(Z,Y).$$

Figure 2.3

where p_i's $(i = 1,2,3)$ are IDB predicates and the relation of p_i is P_i, the s_i's $(i = 1,2)$ are EDB predicates whose relations are $S_1 = \{ab, bc, cd\}$, $S_2 = \{df, fg, gh\}$ (xy is written instead of (x,y)).

If we use one of the above methods then the given result is

$$P_1 = \{ab, bc, cd, ac, bd, ad\} \qquad P_2 = \{df, fg, gh, dg, fh, dh\}$$

$$P_3 = \{af, ag, ah, bf, bg, bh, cf, cg, ch\}$$

This is the least fixed point of **P**, but in the computation there is redundancy of the join operations.

Let us remark that this program is not linear by our Definition 1.5, but it is linear by the Definition 8.9 of [3]. Therefore if we apply the Semi-Naive algorithm [3] for **P** we have P_1, P_2 as the above computed relations, but $P_3 = \{cf, bg, ah\}$ which is not the least fixed point of **P**.

3. The efficient Semi-Naive algorithm

Consider the expression EVAL-RULE-INCR in the Semi-Naive algorithm. This expression is the union of expressions

$$\text{EVAL} - \text{RULE}(r, P_1^{(k)}, \ldots, P_{j-1}^{(k)}, \Delta P_j^{(k)}, P_{j+1}^{(k)}, \ldots, P_\beta^{(k)}); \quad (j = \overline{1, \beta}),$$

where

$$\Delta P_j^{(k)} = P_j^{(k)} - P_j^{(k-1)}.$$

In the algorithm the differences $\Delta P_1^{(k+1)}, \ldots, \Delta P_{i-1}^{(k+1)}$ have not been used to compute $\Delta P_i^{(k+1)}$ even they are computed now. Moreover, the tuples produced from $\Delta P_1^{(k)}, \Delta P_2^{(k)}, \ldots, \Delta P_\beta^{(k)}$ $(k = 0, 1, 2, \ldots)$ will be repeated a lot of times in computing EVAL-INCR-RULE$(r, P_1^{(k)}, \ldots, P_\beta^{(k)}, \Delta P_1^{(k)}, \ldots, \Delta P_\beta^{(k)})$.

It can be modified by substituting the expression

$$\Delta(i, k) = \text{EVAL} - \text{INCR}(p_i, P_1^{(k)}, \ldots, P_\beta^{(k)}, \Delta P_1^{(k)}, \ldots, \Delta P_\beta^{(k)})$$

by the expression

$$\Delta^*(i, k) = \text{EVAL} - \text{INCR}^*(p_i, P_1^{(k)}, \ldots, P_\beta^{(k)}, \Delta P_1^{(k)}, \ldots, \Delta P_\beta^{(k)}) =$$
$$= \bigcup_{1 \le l \le \beta} \text{EVAL}(p_i, T_1^{(k)}, \ldots, T_{l-1}^{(k)}, \Delta Q_l^{(k)}, Q_{l+1}^{(k)}, \ldots, Q_\beta^{(k)}),$$

where

$$Q_j^{(k)} = \begin{cases} P_j^{(k+1)} & \text{if} \quad 1 \le j < i, \\ P_j^{(k)} & \text{if} \quad i \le j \le \beta; \end{cases}$$

$$\Delta Q_j^{(k)} = \begin{cases} \Delta P_j^{(k+1)} & \text{if} \quad 1 \le j < i, \\ \Delta P_j^{(k)} & \text{if} \quad i \le j \le \beta; \end{cases}$$

$$T_j^{(k)} = Q_j^{(k)} - \Delta Q_j^{(k)}$$

for every $k = 0, 1, 2, \ldots$, $i = \overline{1, \beta}$, $j = \overline{1, \beta}$.

Lemma 3.1.

$$\Delta(i, k) \subseteq \Delta^*(i, k).$$

Proof. Suppose that $x \in \Delta(i, k)$, then x set up from $x_l \in P_l^{(k)}$ $(l \neq j)$ and $x_j \in \Delta P_j^{(k)}$ by EVAL-RULE$(r, P_1^{(k)}, \ldots, P_{j-1}^{(k)}, \Delta P_j^{(k)}, P_{j+1}^{(k)}, \ldots, P_\beta^{(k)})$ for some $r \in A_i$ and some j $(1 \le j \le \beta)$, where A_i is the set of rules having the head predicate p_i.

It is clear that $x_l \in P_l^{(k)}$ iff either $x_l \in P_l^{(k-1)}$ or $x_l \in \Delta P_l^{(k)}$.

If $x_1 \in \Delta P_1^{(k)}$ then $x \in$EVAL-RULE$(r, \Delta P_1^{(k)}, P_2^{(k)}, \ldots, P_\beta^{(k)})$ and therefore $x \in \Delta^*(i, k)$ else

if $x_2 \in \Delta P_2^{(k)}$ then $x \in$EVAL-RULE$(r, P_1^{(k-1)}, \Delta P_2^{(k)}, \ldots, P_\beta^{(k)})$, therefore $x \in \Delta^*(i, k)$ else ....

If $x_{j-1} \in \Delta P_{j-1}^{(k)}$ then

$x \in$EVAL-RULE$(r, P_1^{(k-1)}, \ldots, P_{j-2}^{(k-1)}, \Delta P_{j-1}^{(k)}, P_j^{(k)}, \ldots, P_\beta^{(k)})$, therefore $x \in \Delta^*(i, k)$ else $x \in$EVAL-RULE$(r, P_1^{(k-1)}, \ldots, P_{j-1}^{(k-1)}, \Delta P_j^{(k)}, \Delta P_{j+1}^{(k)}, \ldots, P_\beta^{(k)})$ and so $x \in \Delta^*(i, k)$.

Lemma 3.2.

$$\text{EVAL}(p_i, P_1^{(k)}, \ldots, P_\beta^{(k)}) \subseteq P_i^{(k)} \cup \Delta^*(i, k)$$

for every $k = 1, 2, \ldots$, $i = \overline{1, \beta}$.

Proof. It is clear that

$$\text{EVAL}(p_i, P_1^{(k)}, \ldots, P_\beta^{(k)}) = P_i^{(k)} \cup \Delta(i,k) \subseteq P_i^{(k)} \cup \Delta^*(i,k).$$

Now let us describe the details of the modification.

Algorithm 3.1. *Modification of the Semi-Naive algorithm to compute the least fixed point of a Datalog program.*

Input: A collection of the safe Datalog rules with the EDB predicates $r_1, r_2, \ldots, r_\alpha$, the IDB predicates $p_1, p_2, \ldots, p_\beta$ and a list of $R_1, R_2, \ldots, R_\alpha$ to serve as values of the EDB predicates.

Output: The least fixed point solution to the relational equations obtained from these rules.

Method: We use the function EVAL once to get the initial state for the IDB relations and then EVAL-INCR* is used repeatedly on the incremental IDB relations. The computation is shown in Figure 3.1.

```
for i = 1 to β do P_i := ΔP_i := EVAL(p_i, ∅, ..., ∅);
    repeat
        for i = 1 to β do
        ΔQ_i := ∅;
        for j := 1 to β do
            begin if j > 1 then T_{j-1} := P_{j-1} − ΔP_{j-1};
            R := EVAL(p_i, T_1, ..., T_{j-1}, ΔP_j, P_{j+1}, ..., P_β);
            ΔQ_i := ΔQ_i ∪ R
            end;
            ΔP_i := ΔQ_i − P_i;    P_i = P_i ∪ ΔP_i
        end
    until ΔP_i = ∅ for all i;
output P_i's
end
```

Figure 3.1

In the computation for each predicate p_i there is an associated relation P_i which contains all the tuples, and there is an incremental relation ΔP_i which contains only the tuples added in the previous round.

The correctness of the algorithm is shown in the following

Theorem 3.1. *If a Datalog program* **P** *does not contain negation in the bodies and it is rectified to be safe, its EDB database is finite, then Algorithm 3.1 halts and it does so at the least fixed point of the program to which it is applied.*

Proof. It is clear that the sequence $P_i^{(k)}$ $(i = \overline{1,\beta};\ k = 0,1,2,\ldots)$ produced by Algorithm 3.1 is monotonic and the *DOM* of database is finite. Hence, the algorithm halts.

Let P_i' $(i = \overline{1,\beta})$ be the limit of the sequence $P_i'^{(k)}$ produced by the Naive algorithm; P_i'' $(i = \overline{1,\beta})$ be the limit of the sequence $P_i''^{(k)}$ produced by the Semi-Naive algorithm. It is known that $P_i' = P_i''$ $(i = \overline{1,\beta})$.

First we shall show that $P_i''^{(k)} \subseteq P_i^{(k)}$ for every $k = 1,2,\ldots; i = \overline{1,\beta}$.
For $k = 1 : P_i^{(1)} = P_i''^{(1)} = \Delta P_i^{(1)} = \Delta P_i''^{(1)} = \mathrm{EVAL}(p_i, \emptyset, \ldots, \emptyset)$ $(i = \overline{1,\beta})$.

Suppose that for every ℓ $(1 \leq \ell < k)$ $P_i''^{(\ell)} \subseteq P_i^{\ell}$ $(i = \overline{1,\beta})$.
For $k = \ell + 1$:

$$P_i''^{(k)} = P_i''^{(\ell)} \cup \mathrm{EVAL} - \mathrm{INCR}(p_i, P_1''^{(\ell)}, \ldots, P_\beta''^{(\ell)}, \Delta P_1''^{(\ell)}, \ldots, \Delta P_\beta''^{(\ell)}).$$

By $P_i''^{(\ell)} \subseteq P_i^{(\ell)}$ we have $P_i''^{(k)} \subseteq P_i^{(\ell)} \cup \mathrm{EVAL}(p_i, P_1^{(\ell)}, \ldots, P_\beta^{(\ell)}) \subseteq P_i^{(k)}$ $(i = \overline{1,\beta})$.

Next we prove that for each non-negative integer k_2 there exists k_3 such that $P_i^{(k_2)} \subseteq P_i'^{(k_3)}$ for every $i = \overline{1,\beta}$.

Indeed, if $k_2 = 1$ then $k_3 = 1$.

Suppose that for every $\ell_2 < k_2$ there is ℓ_3 such that $P_i^{(\ell_2)} \subseteq P_i'^{(\ell_3)}$ $(i = \overline{1,\beta})$. For $k_2 = \ell_2 + 1$:

$$P_1^{(k_2)} = P_1^{(\ell_2)} \cup \mathrm{EVAL} - \mathrm{INCR}^*(p_1, P_1^{(\ell_2)}, \ldots, P_\beta^{(\ell_2)}, \Delta P_1^{(\ell_2)}, \ldots, \Delta P_\beta^{(\ell_2)}) =$$
$$= P_1^{(\ell_2)} \cup \Delta^*(1, \ell_2),$$

but

$$\Delta^*(1,\ell_2) = \bigcup_{1 \leq j \leq \beta} \mathrm{EVAL}(p_1, P_1^{(\ell_2)}, \ldots, P_{j-1}^{(\ell_2)}, \Delta P_j^{(\ell_2)}, P_{j+1}^{(\ell_2)}, \ldots, P_\beta^{(\ell_2)}),$$

and therefore

$$P_1^{(k_2)} \subseteq P_1'^{(\ell_3+1)}.$$

$$P_2^{(k_2)} = P_2^{(\ell_2)} \cup \mathrm{EVAL} - \mathrm{INCR}^*(p_2, P_1^{(\ell_2)}, \ldots, P_\beta^{(\ell_2)}, \Delta P_1^{(\ell_2)}, \ldots, \Delta P_\beta^{(\ell_2)}) =$$
$$= P_2^{(\ell_2)} \cup \Delta^*(2, \ell_2),$$

but

$$\Delta^*(2,\ell_2) = \mathrm{EVAL}(p_2, \Delta P_1^{(\ell_2+1)}, \ldots, P_\beta^{(\ell_2)}) \cup \bigcup_{2\le j\le\beta} \mathrm{EVAL}(p_2, P_1^{(\ell_2)}, \ldots, P_{j-1}^{(\ell_2)}, \Delta P_j^{(\ell_2)}, P_{j+1}^{(\ell_2)}, \ldots, P_\beta^{(\ell_2)}),$$

and therefore $P_2^{(k_2)} \subseteq P_2^{\prime(\ell_3+2)}$. Analogously, $P_j^{(k_2)} \subseteq P_j^{\prime(\ell_3+j)}$ for every $j = \overline{1,\beta}$ and $P_\beta^{(k_2)} \subseteq P_\beta^{\prime(\ell_3+\beta)}$. By monotonicity of the sequence $P_i^{\prime(k)}$ we have

$$\begin{aligned}
&P_1^{(k_2)} \subseteq P_1^{\prime(\ell_3+1)} \subseteq P_1^{\prime(\ell_3+\beta)},\\
&P_2^{(k_2)} \subseteq P_2^{\prime(\ell_3+2)} \subseteq P_2^{\prime(\ell_3+\beta)},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&P_\beta^{(k_2)} \subseteq P_\beta^{\prime(\ell_3+\beta)}.
\end{aligned}$$

Therefore for $k_3 = \ell_3 + \beta$ we have $P_i^{(k_2)} \subseteq P_i^{\prime(k_3)}$ $(i = \overline{1,\beta})$.

Taking limit of all sequences $P_i^{\prime\prime(k_1)}, P_i^{(k_2)}, P_i^{\prime(k_3)}$ $(i = \overline{1,\beta})$ we have

$$P_i'' \subseteq P_i \subseteq P_i', \quad i = \overline{1,\beta}, \quad \text{but} \quad P_i'' = P_i'.$$

This implies that $P_i = P_i'$ for every $i = \overline{1,\beta}$.

It is not difficult to prove that the number of iterations of Algorithm 3.1 is polynomial in the size of the input of program to which the algorithm is applied. Now we show that the given algorithm converges no slower than the Semi-Naive algorithm.

Theorem 3.2. *For the same program Algorithm 3.1 will halt after no more steps than the Semi-Naive algorithm.*

Proof. In the proof of Theorem 3.1 we can see that $P_i^{\prime\prime(k)} \subseteq P_i^{(k)}$ $(i = \overline{1,\beta})$ for every k.

Moreover, the sequence $P_i^{(k)}$ and the sequence $P_i^{\prime\prime(k)}$ are monotonic and converge to the same limit. Therefore, if $P_i^{\prime\prime(m)} = P_i$ $(i = \overline{1,\beta})$ then $P_i^{(m)} = P_i$, too. This shows that if the Semi-Naive algorithm halts at $k = m$ then Algorithm 3.1 does so at $k \le m$.

Theorem 3.3. *There is no redundancy of join operations in performing Algorithm 3.1.*

Proof. Let r be the rule having the head predicate p_i $(1 \le i \le \beta)$ and k_0 be the least such number that the tuples $x_j \in Q_j^{(k_0)}$ $(j = \overline{1,\beta})$. In the process of

evaluation if there exists one occurence E=EVAL-RULE$(r, \{x_1\}, \ldots, \{x_\beta\})$ in the computing $\Delta^*(i, k_0)$, then it will not be repeated in the computing $\Delta^*(i, k)$ for $k \geq k_0$.

Case a) If $k = k_0$ let j be the least such index that $x_j \in \Delta Q_j^{(k_0)}$ and $x_l \in Q_l^{(k_0-1)}$ for every $1 \leq l < j \leq \beta$. Then E occurs only once in the evaluation of EVAL-RULE$(r, Q_1^{(k_0-1)}, \ldots, Q_{j-1}^{(k_0-1)}, \Delta Q_j^{(k_0)}, \ldots, Q_\beta^{(k_0)})$ for computing $\Delta^*(i, k_0)$.

Case b) If $k > k_0$ then $x_j \notin \Delta Q_j^{(k)}$ $(j = \overline{1, \beta})$, because $\Delta Q_j^{(k)} = Q_j^{(k)} - Q_j^{(k-1)}$ and $k - 1 \geq k_0$.

Finally, we give an example in which case Algorithm 3.1 converges really faster than the Semi-Naive algorithm.

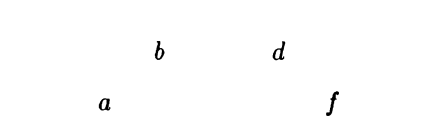
Example 3.1. Given two directed graphs without cycle: $G_1 = (A_1, N_1)$, $G_2 = (A_2, N_2)$, where A_1, A_2 are the sets of nodes, N_1, N_2 are the sets of arcs, $N_1 \cap N_2 = \emptyset$. The program given by Figure 2.3 of Example 2.1 looks for such nodepairs (X, Y), $X \in A_1$, $Y \in A_2$ that there exists a path from X to Y. For each index i $(i = 1, 2)$ let $r_i(X, Y)$ $(i = 1, 2)$ be an EDB predicate, which is true iff there is an arc from X to Y in G_i. p_i is an IDB predicate and $p_i(X, Y)$ $(i = 1, 2)$ is true iff there is a path from X to Y in G_i, $p_3(X, Y)$ is true iff there is a path from $X \in A_1 - A_2$ to $Y \in A_2 - A_1$.

Let R_i $(i = 1, 2)$ be the relation associated to r_i, P_i $(i = 1, 2, 3)$ be the relation associated to p_i. For computing P_3 we have to look for the least fixed point of the system of equations given by Figure 3.2:

$$P_1(X, Y) = R_1(X, Y) \cup \pi_{XY}(P_1(X, Z) \bowtie R_1(Z, Y)),$$
$$P_2(X, Y) = R_2(X, Y) \cup \pi_{XY}(P_2(X, Z) \bowtie R_2(Z, Y)),$$
$$P_3(X, Y) = \pi_{XY}(P_1(X, Z) \bowtie P_2(Z, Y)).$$

Figure 3.2

Now suppose we have graphs G_1 and G_2 shown in Figure 3.3.



$A_1 = \{a, b, c\}$ $N_1 = \{(a, b), (b, c)\}$

$$A_2 = \{c, d, f\} \qquad N_2 = \{(c, d), (d, f)\}.$$

Figure 3.3

For convenience we shall use more compact notations for the tuples, xy instead of (x, y).

Let the EDB relations be $R_1 = \{ab, bc\}$ and $R_2 = \{cd, df\}$.

a) By applying the Semi-Naive algorithm the iteration steps and the solution of equations shown in Figure 3.2 is given by Figure 3.4.

b) By applying the efficient Semi-Naive algorithm the iteration steps and the solution of the same equations is given by Figure 3.5.

step	ΔP_1	P_1	ΔP_2	P_2	ΔP_3	P_3
1	R_1	R_1	R_2	R_2	$\emptyset$	$\emptyset$
2	$\{ac\}$	$R_1 \cup \{ac\}$	$\{cf\}$	$R_2 \cup \{cf\}$	$\{bd\}$	$\{bd\}$
3	$\emptyset$	–	$\emptyset$	–	$\{ad, af, bf\}$	$\{bd, ad, af, bf\}$
4	$\emptyset$	–	$\emptyset$	–	$\emptyset$	–

Figure 3.4

step	ΔP_1	P_1	ΔP_2	P_2	ΔP_3	P_3
1	R_1	R_1	R_2	R_2	$\{bd\}$	$\{bd\}$
2	$\{ac\}$	$R_1 \cup \{ac\}$	$\{cf\}$	$R_2 \cup \{cf\}$	$\{ad, af, bf\}$	$\{ad, bd, af, bf\}$
3	$\emptyset$	–	$\emptyset$	–	$\emptyset$	–

Figure 3.5

4. The stratified evaluation method for finding the least fixed point of a Datalog program

4.1. Connected graph

Now we create a connected graph (C-graph) from the dependency graph (D-graph) for the given Datalog program **P**.

Let $D = (V, N)$ be the D-graph of **P**, where V is the set of oriented arcs and N is the set of nodes (labeling by the names of predicates).

Definition 4.1. We that $a, b \in N$ are connected components of D iff a and b are mutually recursive ($a = b$ is permitted).

Let

$$M_x = \{y : \ x \text{ and } y \text{ are connected components of } D\},$$

$$C(x) = \begin{cases} M_x & \text{if } M_x \neq \emptyset, \\ \{x\} & \text{otherwise.} \end{cases}$$

Notice that if x and y are connected components of D then $M_x = M_y$.

Definition 4.2. The C-graph for **P** is $C = (V', N')$ where

$$N' = \{C(x) : \quad x \in N\},$$
$$V' = \{(\mathbf{u}, \mathbf{v}) : \quad \text{there is } x \in \mathbf{u}, \ y \in \mathbf{v} \ \text{ and } \ (x, y) \in V\}.$$

Because the D-graph of **P** is orientated, the defined C-graph is orientated, too.

For the program **P** given by Example 3.1 the D-graph and the corresponding C-graph are shown in Figure 4.1.

Corollary. *The C-graph does not contain a circle, so we can stratify its nodes. Let* $\mathbf{u} \in N'$, *we say that* **u** *is in stratum* i *if the maximal length of all possible oriented paths to* **u** *is equal to* i.

Let us remark that if a program **P** does not contain mutually recursive predicates then its C-graph is the D-graph.

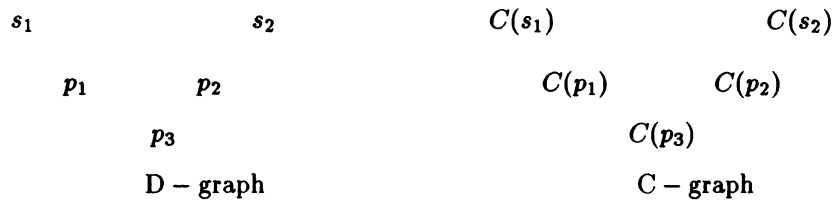


Figure 4.1

We can stratify all the predicates of a given program **P**. If $x \in N$ and $C(x)$ has stratum i, then all predicates in $C(x)$ have stratum i, too. It is clear that every EDB predicate has stratum 0.

Now let us suppose that the predicate p_i is in stratum i and every predicate having stratum $j < i$ is evaluated. Let the program $\mathbf{P}'$ be the set of those rules of **P** whose head predicates are in $C(p)$. The predicates of **P** which have stratum $j < i$ are the EDB predicates of $\mathbf{P}'$.

Next we evaluate the relations for the IDB predicates in $\mathbf{P}'$. If $C(p)$ contains only one predicate and this predicate is non-recursive, then its relation is computed from the EDB relations using relational algebraic operations (EVAL function). Otherwise let the system of equations Π' corresponding to $\mathbf{P}'$ be

$$P_1 = E_1(P_1, \ldots, P_\beta)$$
$$P_2 = E_2(P_1, \ldots, P_\beta)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$P_\beta = E_\beta(P_1, \ldots, P_\beta),$$

where E_i is the relational algebraic expression EVAL corresponding to the predicate p_i.

We compute P_i $(i = \overline{1,\beta})$ as follows:

a) If each equation of Π' is linear with respect to every variable P_i $(i = \overline{1,\beta})$ then we can use the iterations of the Semi-Naive or the Efficient Semi-Naive algorithm.

b) If Π' is linear (see Definition 1.5) then we can apply the following iterations which are given in [3]:

$$P_i^{(k+1)} = P_i^{(k)} \cup \Delta P_i^{(k+1)}; \quad \Delta P_i^{(0)} = P_i^{(0)} = \emptyset \quad (i = \overline{1,\beta};\ k = 0, 1, 2, \ldots),$$

where

$$\Delta P_i^{(k+1)} = E_i(\Delta P_1^{(k)}, \ldots, \Delta P_\beta^{(k)}) - P_i^{(k)}.$$

c) If each equation $P_i = E_i(P_1, \ldots, P_\beta)$ of Π' is linear with respect to P_i then we can use the iterations of the General Semi-Naive algorithm (see Subsection 2.3).

d) If Π' has not the properties a), b), c) then apply the iterations of the Naive algorithm.

4.2. Linearization

In the previous section we gave the Efficient Semi-Naive algorithm for the case when the Datalog program does not contain that rule whose body has repeated occurences of the IDB predicates. To use the above algorithm in general case, we have to transform the given Datalog program by the following way:

Let $r : q :- q_1 \& q_2 \& \ldots \& q_n.$ be the rule which contains repeated occurences of IDB predicates.

Step 1. Look for the least index i such that there is at least two occurences of an IDB predicate q_i.

Step 2. Substitute the rule r by two rules r_1, r_2 using a new predicate symbol p

$$r_1 : \quad q :- q_1 \& \ldots \& q_i \& p.$$

$$r_2 : \quad p :- q_{i+1} \& \ldots \& q_n.$$

If r_2 is linear with respect to every occurence of the IDB predicates in it then exit, else let r be r_2 and go to step 1.

It is easy to see that the rewritten rules are linear with respect to the IDB predicates being in the bodies.

Acknowledgement. The authors would like to thank András Benczur for insightful discussions about this paper and the careful comments on the second part.

References

[1] **Ullman J.D.,** *Principles of Database and Knowledge Base Systems,* Computer Science Press, 1988.

[2] **Gurevich Y. and Shelah S.,** *Fixed-Point Extensions of First Order Logic,* FOCS, 1985.

[3] **Cerl S., Gottlob G. and Tanca L.,** *Logic Programming and Databases,* Springer, 1989.

(Received September 10, 1992)

T. Márkus and Manh Thanh Le
Department of General Computer Science
Eötvös Loránd University
VIII. Múzeum krt. 6-8.
H-1088 Budapest, Hungary