On the Implementation of a Simple Class of Logic Queries for Databases

Domenico Saccà †
CRAI, Rende, Italy

Carlo Zaniolo
MCC, Austin, Texas

1. Introduction

We assume the reader familiar with the basic concepts of *relational databases* [U1] and with the *logical query language* for databases, using PROLOG's notation, described in [U2]. A *database logic query* is expressed as a triple $<G, LP, D>$, where G is a *goal* to be solved using the rules of the *logic program* LP and the facts of the relational database D. In this paper, we study the problem of efficient implementations of queries on recursive rules without function symbols. We focus on an important subclass, called *canonical strongly linear queries (CSL queries)*, and study the *binding-passing property,* which entails the propagation of the initial bindings (established by constants in the query goal) during the top-down (as in backward chaining) execution phase.

The paper is organized as follows. In Section 2, we define CSL queries and study the binding propagation problem. In Section 3, we focus on *1-bound* CSL queries, where the binding propagates to a single (but not always the same) argument of the recursive predicate. In Section 4, we study the problem of implementing these queries. We use a unifying framework to provide a simple description of the following four methods: the *counting* method (informally described in [B+]), the *eager* method (similar to that in [HN]), the *magic set* method (presented in [B+]), and a new method here introduced, called *magic counting,* which combines the advantages of the first and the third. Extensions to and proofs of these results are given in [SZ].

† This work was done while this author was visiting at MCC.

© 1986 ACM-0-89791-179-2/86/0300-0016 $00.75

2. Bound CSL queries

A logic program will be said to be *linear* if it contains at least one recursive rule and every rule has at most one occurrence of a recursive predicate in the body. A query $<G, LP, D>$ is *canonical strongly linear* (*CSL-query*) if LP is linear and contains exactly one recursive rule (hence, exactly one recursive predicate symbol, say R), and the predicate symbol in G and in the head of all rules in LP is R. For simplicity and without loss of generality, we assume that all the arguments of our predicates are variables.

In the following, $Q = <G, LP, D>$ denotes our CSL-query, r_i denotes the recursive rule in LP, and R denotes the recursive predicate of r_i. The *binding-set* of the k-th argument of the head of r_i, denoted by $BS_{r_i}(k)$ is defined as follows:

a) if x is in the k-th argument of the head of r_i then $x \in BS_{r_i}(k)$,

b) if $x \in BS_{r_i}(k)$ and there exists a database predicate in the body of r_i having x and y among its arguments, then $y \in BS_{r_i}(k)$.

LEMMA 1. *If* $BS_{r_i}(k) \cap BS_{r_i}(p) \neq \emptyset$ *then*

$$BS_{r_i}(k) = BS_{r_i}(p). \quad \square$$

We associate to Q the following directed graph $B_Q = <N, A>$ (*binding graph*):

a) $N = \{1, \ldots, n\}$, where n is the arity of the recursive predicate R,

b) $A = \{(j, s) \mid x \in BS_{r_i}(j)$, where x is the variable in the s-th argument of R in the body of $r_i\}$.

We will say that our CSL-query Q is *bound* (alias, it has the *binding-passing* property), when the subgraph of B_Q induced by the set of nodes $\hat{V}$ is cyclic, where $\hat{V}$ is defined as follows:

i) for each s, $1 \leq s \leq n$, if the s-th argument of the goal G is a constant, then s is in $\hat{V}$,

ii) if $j \in \hat{V}$ and (j, s) is in B_Q, then s is in $\hat{V}$.

The binding-passing property guarantees that the initial bindings of the query can be propagated down, via database relations, to any depth of recursion.

Example 1. Consider the following LP (where E and the P's are database predicates):

$$\begin{aligned} r_1:\ R(x_1,x_2,x_3,x_4,x_5,x_6)\text{:-}\ & P_1(x_1,y_1),\ P_2(y_1,\hat{x}_2),\\ & P_3(x_3,\hat{x}_1),\ P_4(x_4,\hat{x}_4),\\ & P_5(x_5,\hat{x}_6),\\ & R(\hat{x}_1,\hat{x}_2,x_2,\hat{x}_4,\hat{x}_5,\hat{x}_6),\\ & P_6(x_6,y_2),\ P_7(\hat{x}_5,y_3).\\ r_2:\ R(x_1,x_2,x_3,x_4,x_5,x_6)\text{:-}\ & E(x_1,x_2,x_3,x_4,x_5,x_6). \end{aligned}$$

The binding graph of BG_Q is shown in Figure 1. The CSL query $Q=<R(a, x_2, x_3, x_4, x_5, x_6), LP, D>$ is bound. On the other hand, $Q=<R(x_1, x_2, x_3, x_4, b, x_6), LP, D>$ is not bound.□

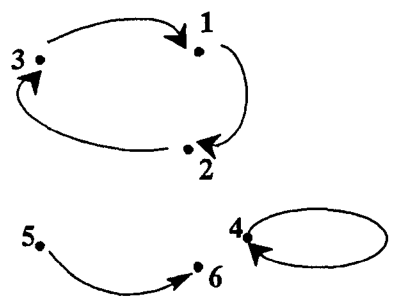

Figure 1. The Binding Graph for Example 1

Given a node i in Q's binding graph B_Q, we denote by $S(i)$ (resp., $A(i)$) the set of all nodes j in B_Q such that (i,j) (resp., (j,i)) is in B_Q). Two nodes i,j are S-equivalent (resp., A-equivalent) if $S(i)=S(j)$ (resp., $A(i)=A(j)$). Being N the nodes of B_Q, let N_0 (resp., $\hat{N}_0$) denote those $i \in N$ s.t. $S(i)$ is empty (resp., $A(i)$ is empty). Then, the partition of $N-N_0$ induced by the S-equivalence will be denoted by $\{N_1, \ldots, N_k\}$ and called the *S-partition* of Q (since Q is bound, $k>0$). Symmetrically, the partition of $N-\hat{N}_0$ induced by the A-equivalence will be denoted $\{\hat{N}_1, \ldots, \hat{N}_{\hat{k}}\}$ and called the *A-partition* of Q. Let now $S(N_q)$ denote the mapping from $\{N_1, \ldots, N_k\}$ to the power set of N, such that $S(Nq)=S(j)$ and j is any node in N_q. Simmetrically, $A(\hat{N}_q)=A(i)$ where i is any node in $\hat{N}_q$. It is then simple to prove that S is a bijection from $\{N_1, \ldots, N_k\}$ to $\{\hat{N}_1, \ldots, \hat{N}_{\hat{k}}\}$, with A its inverse. We will call this the *SA-bijection* of Q.

For Example 1, the S-partition can be listed as follows: $N_1=\{1\}$, $N_2=\{2\}$, $N_3=\{3\}$, $N_4=\{4\}$, $N_5=\{5\}$. The A-partition can be listed as follows: $\hat{N}_1=\{2\}$, $\hat{N}_2=\{3\}$, $\hat{N}_3=\{1\}$, $\hat{N}_4=\{4\}$, $N_5=\{6\}$. Then, the SA-bijection maps N_q into $\hat{N}_q$ for $1 \le q \le 5$.

We note that for each N_q in $\{N_1, \ldots, N_k\}$, $BS_{r_i}(j)=BS_{r_i}(s)$, where r_i is the recursive rule in LP and j,s are any two nodes in N_q. Hence, we can abuse the notation and set $BS_{r_i}(N_q)=BS_{r_i}(j)$, where j is any node in N_q and $1 \le q \le k$. In addition, given a conjunction L (possibly empty) of predicates, we denote by $V[L]$ the set of all variables appearing in all predicates of L. If L is empty then $V[L]=\emptyset$.

THEOREM 1. *Let $Q=<R(m_1, \ldots, m_n), LP, D>$ be our CSL query.. Then the predicates in the body of r_i can be ordered in linear time to yield the following form:*

$$\begin{aligned} r_i:\ R(x_1, \ldots, x_n) \text{:-}\ & L^{N_1 \to S(N_1)}, \ldots, L^{N_k \to S(N_k)},\\ & R(\hat{x}_1, \ldots, \hat{x}_n), W \end{aligned}$$

where $x_1, \ldots, x_n, \hat{x}_1, \ldots, \hat{x}_n$ are variables (not necessarily distinct) and

a) *$L^{N_q \to S(N_q)}$ $(1 \le q \le k)$, is the conjunction (possibly empty) of all database predicates P for which $V[P] \subseteq BS_{r_i}(N_q)$ and W is a conjunction (possibly empty) of all other database predicates in r_i.*

b) *For each $L^{N_q \to S(N_q)}$ $(1 \le q \le k)$, either $L^{N_q \to S(N_q)}$ is empty or $V[L^{N_q \to S(N_q)}]=BS_{r_i}(N_q)$.*

c) *For each q, $1 \le q \le k$, $V[L^{N_q \to S(N_q)}] \cap W=\emptyset$ and for every p, $1 \le p \le k$ and $p \ne q$, $V[L^{N_q \to S(N_q)}] \cap V[L^{N_p \to S(N_p)}]=\emptyset$.*

d) *For each non-empty $L^{N_q \to S(N_q)}$ $(1 \le q \le k)$, $V[L^{N_q \to S(N_q)}] \cap V[R(x_1,..,x_n)]=\{x_j \mid j \in N_q\}$ and $V[L^{N_q \to S(N_q)}] \cap V[R(\hat{x}_1,..,\hat{x}_n)]=\{\hat{x}_j \mid j \in S(N_q)\}$.*

e) *For each empty $L^{N_q \to S(N_q)}$ $(1 \le q \le k)$, $x_j=\hat{x}_s$, where j and s are any two elements of N_q and $S(N_q)$, respectively.*□

For instance, we can label the predicates in r_1 of Example 1 (where the predicates happen to be already in the right order) as follows:

$$\begin{aligned} r_1:\ & R(x_1,x_2,x_3,x_4,x_5,x_6)\text{:-}\\ & P_1^{N_1 \to N_2}(x_1,y_1),\ P_2^{N_1 \to N_2}(y_1,\hat{x}_2),\\ & P_3^{N_3 \to N_1}(x_3,\hat{x}_1),\ P_4^{N_4 \to N_4}(x_4,\hat{x}_4),\\ & P_5^{N_5 \to N_6}(x_5,\hat{x}_6),\ R(\hat{x}_1,\hat{x}_2,x_2,\hat{x}_4,\hat{x}_5,\hat{x}_6),\\ & P_6(x_6,y_2),\ P_7(\hat{x}_5,y_3). \end{aligned}$$

Thus, the predicates in the body of r_1 have been grouped in four non-empty L-conjunctions. The first one (labelled $N_1 \to N_2$), consists of two database predicates (P_1 and P_2). Each of the other three L-conjunctions contains one database predicate, namely, P_3, P_4 and P_5. Notice that the L-conjunction labelled $N_2 \to N_3$ is empty. Finally, P_6 and P_7 end up in the W-conjunction.

Let us now consider the problem of implementing our rules and queries using relational algebra. Queries on non-recursive rules with only database predicates in the body can be implemented by an expression of equijoins (with cartesian products considered a special subcase of these), unions, projects and select operators. For example, if we have the rule

$$r: \ V(x,z) :- \ P(x,y), \ Q(y,z), \ P(w,z),$$

where P and Q are database predicates, then the query $< V(x_1,x_2), \{r\}, \{P,Q\}>$ can be implemented by the following expression $V=\pi_{xz}(P \bowtie Q \bowtie P)$, where P and Q now stand for the *database relations* denoted by the corresponding database predicates in r, and $\bowtie$ corresponds to the natural join of these relations once we regard variables in the predicates as column names of the corresponding relations. A minor complication with this notation of convenience occurs when the same variable appears several times in a predicate. Then, the join must be preceded by a select operator which selects only those tuples having identical values in all identically named columns. This operation, which basically corresponds to unification (since there are no constants and function symbols), will be denoted by ρ. Thus the answer to the previous query when

$$r: \ V(x,z) := \ P(x,y), \ Q(y,z), \ P(z,z)$$

is $V=\pi_{xz}(P \bowtie Q \bowtie \rho(P))$, where ρ here stands for $\sigma_{1=2}$. Since the ρ operator yields relations where equally named columns are identical, any of these can then be used to perform the joins or projects.

The answer to a CSL query can be computed by a fixpoint iteration over a relational algebra expression, and due to the absence of function symbols and the finiteness of the database, the process terminates. Theorem 1, tell us that said relational algebra expression has the following form:

$R= \pi_X(L^{N_1 \to S(N_1)} \bowtie \cdots \bowtie L^{N_k \to S(N_k)} \bowtie \hat{R} \bowtie W) \cup E$ where E denotes the contribution of the non-recursive rules to the fixpoint computation, and $L^{N_t \to S(N_t)}$ and W, which we call *L-joins,* are relations constructed by taking the natural join of the database relations in the corresponding L-conjunctions if these are not empty, and identity relations (over finite database domains) otherwise. For instance in Example 1, $L^{N_1 \to S(N_1)} = P_1 \bowtie P_2$ and $L^{N_3 \to N_1} = P_3^{N_3 \to N_1}$, $L^{N_4 \to N_4} = P_4^{N_4 \to N_4}$ and $L^{N_5 \to N_6} = P_5^{N_5 \to N_6}$ while $L^{N_2 \to N_3}$ is the identity relation defined, for example, over the second column of P_2. Moreover, $W=P_6 \bowtie P_7$.

Let Z and $\hat{Z}$ be two lists of column names and let $T_1, \ldots, T_s$ be relations. Let $T_j \rhd\!\!\lhd_{<Z,\hat{Z}>} T_r$ denote $\pi_{<\overline{T}_j, \overline{T}_r>}(T_j \bowtie T_r)$, where $\overline{T}_j$ (resp., $\overline{T}_r$) is the list of column names of T_j (resp., T_r) which are in $<Z \cup \hat{Z}>$. We denote by $\rhd\!\!\lhd_{[Z,\hat{Z}]}<T_1, \ldots, T_s>$ the following relational algebra expression:

$$\pi_Z(T_1 \rhd\!\!\lhd_{<Z,\hat{Z}>} T_2 \cdots \rhd\!\!\lhd_{<Z,\hat{Z}>} T_s)$$

where the $\rhd\!\!\lhd$ operations are performed in the order they are written (thus, first compute $T_1 \rhd\!\!\lhd_{<Z,\hat{Z}>} T_2$ giving $\ddot{T}_2$, then compute $\ddot{T}_2 \rhd\!\!\lhd_{<Z,\hat{Z}>} T_3$ giving $\ddot{T}_3$, and so on).

We can now state a useful corollary of Theorem 1.

COROLLARY 1.

$$\pi_X(L^{N_1 \to S(N_1)} \bowtie \cdots \bowtie L^{N_k \to S(N_k)} \bowtie \hat{R} \bowtie W) =$$

$$\rhd\!\!\lhd_{[X,\hat{X}]}<L^{N_t \to S(N_t)}, \hat{R}, C^{N_t \to S(N_t)}, W>$$

where $C^{N_t \to S(N_t)}$ *is the list of relations corresponding to all conjunctions* $L^{N_t \to S(N_t)}$ *in the body of the recursive rule* r_i, *which are different from* $L^{N_t \to S(N_t)}$. □

3. Properties of 1-bound CSL queries

Let $Q=<G, LP, D>$ be a CSL-query and let B_Q be the binding graph of Q. Q is *1-bound* if it is bound with only one argument of G constant, and both the indegree and outdegree of every node in B_Q are ≤ 1.

PROPOSITION 1. *Given a 1-bound CSL-query* $Q=<R(m_1,m_2, \ldots, m_q, \ldots, m_n), LP, D>$, *with* m_q *a constant and all other* m_i *variables (not necessarily distinct), then*

a) *each* N_k *in the S-partition and each* $\hat{N}_k$ *in the A-partition are singleton, and*

b) *there exists a non-empty list of singletons* $<N_{j_1}, \ldots, N_{j_p}>$ *(called the active binding cycle of Q) such that they belong to both the S-partition and the A-partition, and* $N_{j_1}=\{q\}$, $S(N_{j_p})=N_{j_1}$ *and for each* $N_{j_s}, 1 \leq s \leq p-1$, $S(N_{j_s})=N_{j_{s+1}}$. □

In [SZ] it is shown that the recognition of whether Q is a 1-bound CSL-query and the construction of its active binding cycle can be done in linear time.

Let us now study the impact of the database D upon a query. The image set of a set T with respect to the L-join $L^{N_h \to S(N_h)}$, denoted $\mathbf{I}^{N_h \to S(N_h)}(T)$, is constructed as follows:

$$\mathbf{I}^{N_h \to S(N_h)}(T)=\pi_{\hat{z}_s}(\sigma_{z_j=T}(L^{N_h \to S(N_h)}))$$

where $N_h=\{j\}$ and $S(N_h)=\{s\}$, and the L-conjunction $L^{N_h \to S(N_h)}$ is not empty. On the other hand, if $L^{N_h \to S(N_h)}$ is empty, by Theorem 1 (part e), we set $\mathbf{I}^{N_h \to S(N_h)}(T)=T$.

The *magic graph* of Q is the directed graph $MG_Q=<M_Q, E_Q>$, where the set of nodes M_Q and the set of arcs E_Q are defined as follows:

a) $[N_{j_1}, a]$, where a denotes the given constant in the q-th argument of the query goal, is in M_Q and it is called the *source node*, moreover

b) if $[N_h, b]$ is in M_Q and c is in $\mathbf{I}^{N_h \to S(N_h)}(\{b\})$ then $[S(N_h), c]$ is in M_Q and $([N_h, b], [S(N_h), c])$ is in E_Q.

The magic graph is a p-partite graph, where p is the size of the active binding cycle, and it can be constructed in linear time.

For each N_h in the active binding cycle, let $M_Q^{N_h}$ denote all the nodes in M_Q that have N_h as first component. $M_Q^{N_h}$ will be called a *magic set.*

A 1-bound CSL query Q will be said to be *acyclic (cyclic)* when its magic graph MG_Q is acyclic (cyclic). Q will be called *regular* if it is acyclic and for each pair of nodes s, j in M_Q, all directed paths from s to j have the same length.

FACT 1. *The collection of non-empty magic sets is a partition of the nodes of the magic graph. Furthermore, if some magic set is empty then the query is regular.* □

For the LP of Example 1, the active binding cycle of $Q = <R(a, x_2, \cdots, x_6), LP, D>$ is the cycle $<N_1, N_2, N_3>$ of Figure 1. Let D be:

$$P_1 = \{(a, b_1), (a_3, b_2), (b_3, b_4)\}$$

$$P_2 = \{(b_1, a_1), (b_1, a_2), (b_2, a_3)\}$$

$$P_3 = \{(a_1, a_3), (a_2, a_3), (a_3, a_4)\}$$

Then, the magic graph of Q is that of Figure 2. We note that Q is regular. If we replace the tuple (a_3, a_4) in P_3 by (a_3, a) then Q becomes cyclic, whereas if we add the tuple (a_1, a_4) to P_3 then Q becomes acyclic but not regular.

We now define another covering (possibly infinite) of the nodes of a magic graph. Each element of this covering contains all nodes which have the same distance from the source node (note that all nodes are reachable from the source node). It turns out that the same node may appear in many elements of this covering.

Let $M_Q^1 = \{[N_{j_1}, a]\}$. For each t, $t > 1$, let $M_Q^t = \{j \mid j \in M_Q$ and there is a (possibly cyclic) path from $[N_{j_1}, a]$ to j of length $t-1\}$: Furthermore, for each t, $t \geq 1$, if M_Q^t is not empty then M_Q^t will be called a *counting set.* Obviously, if the magic graph is cyclic then there are infinite counting sets.

FACT 2. *The counting sets are a covering of the nodes of the magic graph. Furthermore, if M_Q^t is a counting set then $M_Q^t \subseteq M_Q^{N_h}$, where $h = ((t-1)$ **mod** $p) + 1$.* □

Fact 2 says that all nodes in a counting set have the same first component, which is moreover uniquely determined once the index t is given. This means that we can drop this redundant first component and regard counting sets as being sets of database values. Having made this convention for the rest of the paper, we can state the following property:

FACT 3. *For each t, $t > 1$, $M_Q^t = \mathrm{I}^{N_h \to S(N_h)}(M_Q^{t-1})$, where $h = ((t-1)$ **mod** $p) + 1$.*

The *depth* of the magic graph is defined as the maximum length of directed paths in MG_Q.

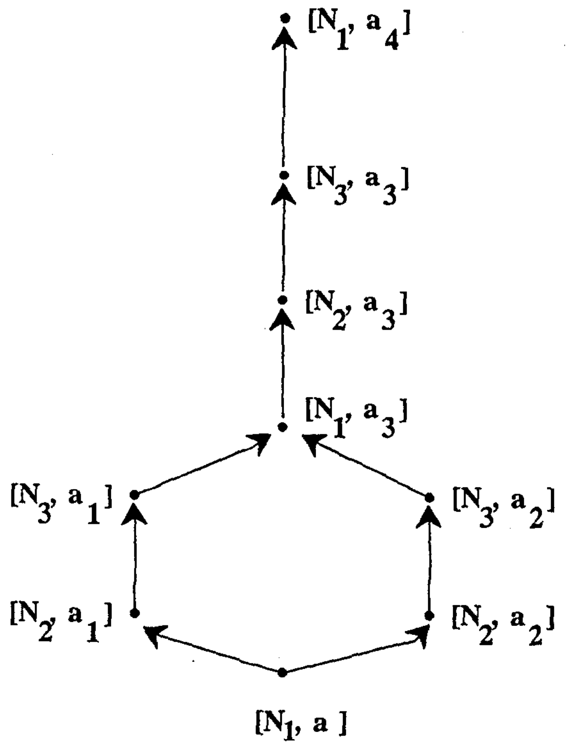

Figure 2. A Regular Magic Graph

PROPOSITION 2. *The counting sets of an acyclic 1-bound CSL-query Q can be computed in $O(e^2)$ time, where e is the total number of arcs in the magic graph of Q. Furthermore, the number of counting sets is $t+1$, where t is the depth of the magic graph.* □

PROPOSITION 3. *Let Q be a 1-bound CSL-query. The counting sets of Q are a partition of the nodes of the magic graph MG_Q if and only if Q is regular. Furthermore, if Q is regular then the counting sets can be computed in $O(e)$ time, where e is the number of arcs in magic graph of Q.* □

4. Methods for implementing 1-bound CSL queries

Let $Q = <R(m_1, m_2, \ldots, m_q, \ldots, m_n), LP, D>$ be a 1-bound CSL-query, where m_q is a constant (say a), whereas all other m_i are variables (not necessarily distinct). Because of Proposition 1, we can assume, without loss of generality, that $<N_1, N_2, \ldots, N_{p-1}, N_p>$ is the active binding cycle of Q, with $N_1 = \{q\}$, $N_2 = S(N_1)$, $\ldots$, $N_p = S(N_{p-1})$, $N_1 = S(N_p)$; also $\hat{N}_1 = S(N_p) = N_1$, $\hat{N}_2 = S(N_1) = N_2$, $\ldots$, $\hat{N}_p = S(N_{p-1}) = N_p$. Thus, by Theorem 1, our recursive rule r_i can be written as follows:

$$
\begin{aligned}
r_i:\ & R(x_1, x_2, \ldots, x_n) :- \\
& L^{N_1 \to N_2}, L^{N_2 \to N_3}, \ldots, L^{N_p \to N_1}, \\
& L^{N_{p+1} \to S(N_{p+1})}, \ldots, L^{N_k \to S(N_k)}, \\
& R(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n), W
\end{aligned}
$$

where $x_1, \ldots, x_n, \hat{x}_1, \ldots, \hat{x}_n$ are variables (not necessarily distinct) and $L^{N_1 \to N_2}, \ldots, L^{N_k \to S(N_k)}$, and W are conjunctions (possibly empty) of database predicates which have the properties stated by Theorem 1. Because of the definition of CSL queries, all other rules in LP are non-recursive and have the form:

$$R^j(z_1, z_2, \ldots, z_n) :- E^j \qquad (3.1)$$

where $z_1, z_2, \ldots, z_n$ are variables (not necessarily distinct), and E^j is a non-empty conjunction of database predicates.

Let us now solve our query Q using backward chaining (*top-down*) execution. We have to compute

$$Answer(Q) = \sigma_{(m_q = a)}(\hat{R}_1)$$

Thus the answer to our query can be computed once we have relation $\hat{R}_1$. In turn, $\hat{R}_1$ can be computed as follows:

$$
\begin{aligned}
\hat{R}_1 &= \rho R_1 \\
R_1 &= \pi_X(L^{N_1 \to N_2} \bowtie \cdots \bowtie \hat{R}_2 \bowtie W) \cup E
\end{aligned}
$$

where E is the union of relational algebra expression that implements all the non-recursive rules (3.1), and ρ is the select operator which select tuples where identically named columns are identical.

Therefore, the computation of $\hat{R}_1$ leads to the evaluation of $\hat{R}_2$, and this lead to the evaluation of $\hat{R}_3$, and so on. Using Corollary 1, we can write this top-down evaluation sequence as follows:

$$
\begin{aligned}
Answer(Q) &= \sigma_{(m_q = a)}(\hat{R}_1) \\
\hat{R}_1 &= \rho(R_1) \\
R_1 &= \ltimes_{[X, \hat{X}]} < L^{N_1 \to N_2}, \hat{R}_2, C^{N_1 \to N_2}, W > \cup E \\
\hat{R}_2 &= \rho(R_2) \\
R_2 &= \ltimes_{[X, \hat{X}]} < L^{N_2 \to N_3}, \hat{R}_3, C^{N_2 \to N_3}, W > \cup E \\
& \ldots \\
\hat{R}_p &= \rho(R_p) \\
R_p &= \ltimes_{[X, \hat{X}]} < L^{N_p \to N_1}, \hat{R}_{p+1}, C^{N_p \to N_1}, W > \cup E \\
\hat{R}_{p+1} &= \rho(R_{p+1}) \\
R_{p+1} &= \ltimes_{[X, \hat{X}]} < L^{N_1 \to N_2}, \hat{R}_{p+2}, C^{N_1 \to N_2}, W > \cup E \\
& \ldots \\
\hat{R}_t &= \rho(R_t) \\
R_t &= \ltimes_{[X, \hat{X}]} < L^{N_h \to S(N_h)}, \hat{R}_{t+1}, C^{N_h \to S(N_h)}, W > \cup E \\
& \ldots
\end{aligned}
$$

where $h = ((t-1) \textbf{ mod } p) + 1$ and p is the size of the active binding cycle. Furthermore, $X = < x_1, \ldots, x_n >$ and $\hat{X} = < \hat{x}_1, \ldots, \hat{x}_n >$.

$Answer(Q)$ is the limit of the following computation: for each level s, set $\hat{R}_{s+1} = \emptyset$ and compute $Answer^s(Q)$ using the first $2 \times s + 1$ equations. Then, we obtain the sequence $Answer^1(Q)$, $Answer^2(Q)$, ..., where $Answer^s(Q) \subseteq Answer^{s+1}(Q)$, and the limit is $Answer(Q)$. But $Answer(Q)$ is finite; therefore, for some t, $Answer(Q) = Answer^t(Q)$. Hence, our goal is to find such a t. To this end, we start by propagating downward the initial binding provided by the selection on $(m_q = a)$. This means that the expression for computing R_1 can be replaced by

$$R_1 = \sigma_{(x_q = a)}(\ltimes_{[X, \hat{X}]} < L^{N_1 \to N_2}, \hat{R}_2, C^{N_1 \to N_2}, W > \cup E).$$

since the column x_q of R_1 corresponds to the column m_q of $\hat{R}_1$.

Let us now denote by x_{N_h} the element of a singleton set N_h in the active binding cycle (thus, x_{N_1} will denote x_q, since $N_1 = \{q\}$). Moreover, we can further propagate the initial binding by noting that the variable x_{N_1} is not in $C^{N_1 \to N_2}$ nor in W by (Theorem 1, part c) (i.e., the corresponding relations do not have a column named x_{N_1}). Hence, we can write

$$
\begin{aligned}
R_1 = & \ltimes_{[X, \hat{X}]} < \sigma_{(x_{N_1} = a)}(L^{N_1 \to N_2}), \hat{R}_2, C^{N_1 \to N_2}, W > \\
& \cup \sigma_{(x_{N_1} = a)}(E)
\end{aligned}
$$

Again by Theorem 1 (part d), both $L^{N_1 \to N_2}$ and $\hat{R}_2$ have a column named $\hat{x}_{N_2}$, whereas all other columns of the two relations have different names. It follows that we can propagate the selection to the expression for computing R_2 using $\pi_{\hat{x}_{N_2}}(\sigma_{(x_{N_1} = a)}(L^{N_1 \to N_2}))$, and so on.

By Fact 3, $M_Q^1 = \{a\}$ and $M_Q^2 = \mathbf{I}^{N_1 \to N_2}(M_Q^1) = \pi_{\hat{x}_{N_2}}(\sigma_{(x_{N_1} = \{a\})}(L^{N_1 \to N_2}))$. Hence, we can write:

$$
\begin{aligned}
Answer(Q) &= \hat{R}_1 \\
\hat{R}_1 &= \rho(R_1) \\
R_1 &= \ltimes_{[X, \hat{X}]} < \sigma_{(x_{N_1} = M_Q^1)}(L^{N_1 \to N_2}), \hat{R}_2, C^{N_1 \to N_2}, W > \\
& \quad \cup \sigma_{(x_{N_1} = M_Q^1)}(E) \\
\hat{R}_2 &= \rho(R_2) \\
R_2 &= \ltimes_{[X, \hat{X}]} < \sigma_{(x_{N_2} = M_Q^2)}(L^{N_2 \to N_3}), \hat{R}_3, C^{N_2 \to N_3}, W > \\
& \quad \cup \sigma_{(x_{N_2} = M_Q^2)}(E) \\
& \quad \ldots \\
\hat{R}_p &= \rho(R_p) \\
R_p &= \ltimes_{[X, \hat{X}]} < \sigma_{(x_{N_p} = M_Q^p)}(L^{N_p \to N_1}), \hat{R}_{p+1}, C^{N_p \to N_1}, W > \\
& \quad \cup \sigma_{(x_{N_p} = M_Q^p)}(E)
\end{aligned}
$$

$\hat{R}_{p+1} = \rho(R_{p+1})$

$R_{p+1} = \ltimes_{[X,\hat{X}]} < \sigma_{(x_{N_1}=M_Q^{p+1})}(L^{N_1 \rightarrow N_2}), \hat{R}_{p+2}, C^{N_1 \rightarrow N_2}, W >$

$\quad \cup\, \sigma_{(x_{N_1}=M_Q^{p+1})}(E)$

$\quad \ldots$

$\hat{R}_t = \rho(R_t)$

$R_t = \ltimes_{[X,\hat{X}]} < \sigma_{(x_{N_h}=M_Q^t)}(L^{N_h \rightarrow S(N_h)}), \hat{R}_{t+1}, C^{N_h \rightarrow S(N_h)}, W >$

$\quad \cup\, \sigma_{(x_{N_h}=M_Q^t)}(E)$

$\quad \ldots$

Notice that if $L^{N_h \rightarrow S(N_h)}$ happens to be an empty conjunction then we can assume that the selection $\sigma_{(x_{N_h}=M_Q^t)}$ is applied to $\hat{R}_{t+1}$ since $x_{N_h} = \hat{x}_{S(N_h)}$ (Theorem 1 part e).

If the query Q is acyclic then the counting sets are finite and, then, there is an M_Q^{t+1} which is empty. Therefore, $\hat{R}_{t+1} = \emptyset$. Hence, we have that

$$R_t = \sigma_{(x_{N_h}=M_Q^t)}(E)$$

Then we can compute $R_{t-1}, \ldots,$ *Answer* (Q) by solving the expressions of the first t levels. However, since we already know that the q-th column of the result only contains the value a, we can use a more efficient method which returns as result the projection of $\hat{R}_1$ on all columns but the q-th. An informal description of this method (called the *counting* method) can be found in [B+]. Let us now present an algorithm for this method. (In the algorithm shown below, by $\pi[-x_j]$ we denote the projection on all columns but those named x_j).

Counting Algorithm

```
1) Compute the counting sets $M_Q^1, \ldots, M_Q^t$ and
   set $h = ((t-1)$ mod $p) + 1$.
2) Set $R_t = \pi_{[-x_{N_h}]}(\sigma_{(x_{N_h}=M_Q^t)}(E))$.
3) for $v = t, t-1, \ldots, 2$ do
   begin
      4) Set $\hat{R}_v = \rho(R_v)$.
      5) Set $w = v-1$ and $g = ((w-1)$ mod $p) + 1$.
      6) Compute $R_w$ as
         $\pi_{[-x_{N_g}]}(\ltimes_{[X,\hat{X}]} < \hat{R}_v, C^{N_g \rightarrow S(N_g)}, W >$
         $\cup\, \sigma_{x_{N_g}=M_Q^v}(E))$
      7) Set $h = g$.
   end
8) Set Answer $(Q) = \rho(R_1)$.
```

After observing that portions of the answer can be computed while constructing the counting sets, we obtain a new algorithm (called the *eager* method) which is similar to that presented in [HN]. (In the algorithm shown below, t represents the number of counting sets).

Eager Algorithm

```
1) Set $M_Q^1 = \{N_1, a\}$,
2) Set Answer $(Q) = \pi_{[-x_{N_1}]}(\sigma_{(x_{N_1}=M_Q^1)}(E))$.
3) for $u = 2, \ldots, t$ do
   begin
      4) Set $h = ((u-1)$ mod $p) + 1$ and
         $M_Q^u = I^{N_h \rightarrow S(N_h)}(M_Q^{u-1})$
      5) Set $R_u = \pi_{[-x_{N_h}]}(\sigma_{(x_{N_h}=M_Q^u)}(E))$.
      6) for $v = u, u-1, \ldots, 2$ do
         begin
            7) Set $\hat{R}_v = \rho(R_v)$.
            8) Set $w = v-1$ and $g = ((w-1)$ mod $p) + 1$.
            9) Compute $R_w$ as
               $\pi_{[-x_{N_g}]}(\ltimes_{[X,\hat{X}]} < \hat{R}, C^{N_g \rightarrow S(N_g)}, W >)$
            10) Set $h = g$.
         end
      11) Set Answer $(Q)$ = Answer $(Q) \cup \rho(R_1)$.
   end
```

We now have the following result concerning correctness (defined as the property that the given procedure terminates and produces all the answers to the given query [VK]).

THEOREM 2 *The counting method and the eager method are correct with respect to a 1-bound CSL query Q if and only if Q is acyclic.* □

It can also be shown that the counting method works better (in terms of tuples retrieved) than the eager method for all acyclic 1-bound CSL queries. On the other hand, the eager method is more storage efficient since only the current counting set needs to be kept. However, as it will be shown later in this section, keeping all counting sets is very important for checking the termination condition for cyclic queries.

The *magic set* method [B+] is described next. In [B+], it is assumed that the active binding cycle is given as input. In our case, all is needed is the query. In this method the results are computed bottom-up using, at each level, the magic set M_Q^N instead of the corresponding counting set $M_Q^t \subseteq M_Q^N$. Once the value of R remains unchanged for an entire cycle, the iteration stops and the result is computed by means of a selection.

Magic Set Algorithm

```
1) Compute the magic sets $M_Q^{N_1}, \ldots, M_Q^{N_p}$, and
   set $h = 1$ $\hat{N} = N_1$, cycle $= 1$, end $=$ false.

2) Set $R = R_1 = \sigma_{(x_{N_h}=M_Q^N)}(E)$.
```

```
3)  while $end = false$ do
    begin
    4)  Set $\hat{R} = \rho(R)$.
    5)  Set $N = A(\hat{N})$.
    6)  Compute $R$ as
        $\ltimes_{[X,\hat{X}]} < \sigma_{(x_N = M_Q^N)}(L^{N \to \hat{N}}), \hat{R}, C^{N \to \hat{N}}, W >$
        $\cup\ \sigma_{(x_N = M_Q^N)}(E)$
    7)  if $cycle < p$ then
        8)  Set $cycle = cycle + 1$.
        else
        9)  if $R = R_1$ then set $end = true$
            else set $cycle = 1$ and $R_1 = R$.
    10) Set $\hat{N} = N$.
    end
11) Set $Answer(Q) = \sigma_{(m_t = a)}(\rho(R))$.
```

THEOREM 3. *The magic set method is correct with respect to all 1-bound CSL queries.* □

Performance-wise neither the Counting Method nor the Magic Set Method is superior in all cases, since the former applies a sharper selection at the various steps of the bottom-up computation than the latter, but computing all counting sets may be more expensive (because of duplicates) than computing the magic sets. However, when the query is regular, by Proposition 4, the counting sets can be computed as efficiently as the magic sets. Thus, we have the following result.

THEOREM 4. *If P and $\hat{P}$ denote the numbers of database tuples respectively retrieved by the counting method and the magic set method in a regular 1-bound CSL-query, then, $P \le \hat{P} + O(t)$, where t is the depth of the magic graph.* □

The $O(t)$ possible loss of performance of the Counting Method versus the Magic Set Method can be considered negligible in view of the fact that the computation of magic sets requires at least $O(t)$ time. On the other side, it may happen that $\hat{P} = P + r$ and r is of some order of magnitude greater than t. Therefore, it is reasonable to assume that the counting method works better than the magic set method for regular queries. Unfortunately, we do not want to construct the magic graph before deciding which is the best method to apply. Instead, we can use a new and and efficient method, called the *magic counting* method, which is correct with respect to all 1-bound CSL queries and coincides with the counting method when the query is regular.

The magic counting method starts by computing tne counting sets, but once it detects that some part of the magic graph is non-regular, it also it removes duplicate nodes. Eventually, some counting sets will be used to construct subsets of the magic sets. Thus, the method constructs a subclass C of the counting sets and a class M of smaller magic sets such that $C \cup M$ is a partition of the nodes of the magic graph. The *magic counting sets* are the elements of $C \cup M$, where $C = \{M_Q^u \mid M_Q^u$ is a counting set and for each counting set $M_Q^v, v > u$, $M_Q^u \cap M_Q^v = \emptyset\}$, and

$$M = \{\hat{M}_Q^N \mid \hat{M}_Q^N \neq \emptyset \text{ and } \hat{M}_Q^N = M_Q^N - (\bigcup_{M_Q^u \in C} M_Q^u)\}.$$

PROPOSITION 4. *The magic counting sets are a partition of the nodes of the magic graph and can be computed in $O(e)$ time, where e is the number of arcs in the magic graph.* □

We are now ready to present the algorithm of the magic counting method.

Magic Counting Algorithm

```
1)  Compute the magic counting sets in $C \cup M$.
2)  If $M \neq \emptyset$ then
    begin
    3)  Perform Steps 3 - 10 of the algorithm of the
        Magic Set Method
    4)  If $|C| = 0$ then
        5)  Set $Answer(Q) = \sigma_{(m_t = a)}(\rho(R))$.
    6)  else Set $E = E \cup R$.
    end
7)  If $|C| > 0$ then
    begin
    8)  Set $t = |C|$
    9)  Perform Steps 2 - 8 of the algorithm of the
        Counting Method.
    end
```

THEOREM 5. *Let Q be a 1-bound CSL-query.*

a) *The magic counting method is correct with respect to Q.*

b) *If P and $\hat{P}$ respectively denote the the numbers of database tuples retrieved by the magic counting method and the magic set method, then for a 1-bound CSL query, $P \le \hat{P} + O(c)$, where c is the number of the magic counting sets that are also counting sets.*

c) *If Q is regular then the magic counting method coincides with the counting method.* □

It turns out that the magic counting method works better than the other methods in most cases. Actually, it could work worse than the eager method or the counting method only if the query is acyclic but not regular. But then, to guarantee termination, one must check that there is no cycle in the database; unfortunately, it not easy to distinguish non-regularity from cyclicity (the transitive closure of all nodes in the magic graph must be computed, whereas all methods compute only the closure of the source node). Therefore, the magic counting method appears to be the best all-around algorithm.

Example 2. Consider the following 1-bound CSL-query $Q = <R(a,y), LP, D>$, where LP is

$$r_1\colon\ R(x,y) :\!-\ E(x,y).$$

$$r_2\colon\ R(x,y) :\!-\ L(x,x_1),\ R(x_1,y_1),\ W(y_1,y).$$

and D is one of the following three databases (this example is taken from [B+]):

Case a):

(a,b_i) and (b_i,c) $(1 \le i \le n)$ are in the relation L, (c,d) is in E, (d,e_i) and (e_i,f) $(1 \le i \le n)$ are in W, and L, W and E do not contain other tuples.

Case b):

(a_i,a_{i+1}) $(1 \le i \le n)$ and (a_1,a_i) $(3 \le i \le n)$ are in the relation L, (a_n,b_n) is in E, (b_i,b_{i-1}) $(2 \le i \le n)$ are in W, and L, W and E do not contain other tuples.

Case c):

(a_i,a_{i+1}) $(1 \le i \le n)$ are in the relation L, (a_i,b_i) $(2 \le i \le n)$ are in E, (b_i,b_{i-1}) $(2 \le i \le n)$ are in W, and L, W and E do not contain other tuples.

It is easy to see that the performances of the methods described in this paper with respect to Q are:

METHOD	CASES a	b	c
Counting	$O(n)$	$O(n^2)$	$O(n)$
Eager	$O(n)$	$O(n^2)$	$O(n^2)$
Magic Set	$O(n^2)$	$O(n)$	$O(n)$
Magic Counting	$O(n)$	$O(n)$	$O(n)$

Notice that Q is regular in the cases a) and c), whereas it is acyclic but not regular in the case b). □

ACKNOWLEDGEMENT.
We wish to thank François Bancilhon for many helpful discussions and comments.

5. References

[B] Bancilhon, F., "A note on the performance of Rule Based Systems", MCC Technical Report, 1985.

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

[CH] Chandra, A.K., Harel, D., "Horn clauses and the fixpoint hierarchy", *Proc. ACM SIGMOD-SIGACT Symp. on Principles of Database Systems*, 1982, pp. 158-163.

[HN] Henschen, L.J., Naqvi, S. A., "On compiling queries in recursive first-order databases", *JACM 31*, 1, 1984, pp. 47-85.

[SZ] Saccà, D., Zaniolo, C., "Implementation of strongly linear logic queries for databases", unpublished manuscript, 1985.

[U1] Ullman, J.D., *Principles of Database Systems*, Computer Science Press, Rockville, Md., 1982.

[U2] Ullman, J.D., "Implementation of logical query languages for databases", *TODS 10*, 3, 1985, pp. 289-321.

[VK] van Emden, M.H., Kowalski, R., "The semantics of predicate logic as a programming language", *JACM 23*, 4, 1976, pp. 733-742.