PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT. © ICOT, 1988

Efficient Query Answering on Stratified Databases

Jean-Marc KERISIT, Jean-Marc PUGIN

Bull CEDIAG (A.I. Corporate Center), Bull Research Center
68, route de Versailles, 78430 Louveciennes FRANCE

ABSTRACT

We propose an extension to two deduction methods (Alexander Method and Magic Sets) to treat queries on stratified databases (Apt et al. 1986). As shown in (Balbin et al. 1987), incorporating NAF-like rules to the original algorithms may produce unstratified programs. Instead of modifying the algorithms in order to obtain only stratified programs, we show that the transformed programs belong to a new class (the so-called weakly stratified programs) for which we define a fixed-point semantics by means of a particular model (W-model). We first prove that, in the case of stratified programs, the W-model is exactly the 'natural model' (Apt et al. 1986). Then we show soundness and completeness of our method: for a given query to a stratified program, the W-model of the transformed program computes exactly the set of answers to this query. Our work is more efficient than the earlier work dealing with the same topic (Balbin et al. 1987), essentially because it avoids data duplication.

INTRODUCTION

Deductive Databases viewed as logic programs have two components: the Extensional Database (EDB), a set of ground function-free unit clauses which represent tuples in relations, and the Intensional Database (IDB), a set of function-free clauses with predefined variables which represent definitions for relational views (Gallaire and Minker 1978). Our aim is to define a query evaluation system able to compute efficiently answers to queries on views by using their (eventually recursive) definition in IDB.

The case when IDB is a set of Horn clauses has been the subject of numerous works (see (Bancilhon and Ramakrishnan 1986) and (Demolombe and Royer 1986) for a survey) which have led to different solutions: the Alexander Method (Rohmer et al. 1986), APEX (Kifer and Lozinskii 1986a), Counting (Sacca and Zaniolo 1986), Magic functions (Gardarin and de Maindreville 1987), Magic Sets (Bancilhon et al. 1986), QSQ (Vieille 1986),...

When negation is allowed within the body of clauses, two problems arise: define the "natural" semantics of logic programs and propose an efficient query evaluation system compatible with this semantics.

Concerning the first point, we refer here to an interesting framework which has been introduced simultaneously by Apt et al. (1986) and van Gelder (1986): *Stratification.* The principle of Stratification consists in disallowing recursion "through" Negation. For a given DDB, this property is guaranteed by a pure syntactic condition based on the decomposition of the DDB into ordered *strata.* Then, its meaning is properly defined by its *Natural Model* which can be computed by iterations of fixed-point operators on the strata.

As for the second issue, our approach is to extend a classical query evaluation system to treat Stratified Databases. With this end in view, we propose an extension both to the *Alexander Method* (Rohmer et al. 1986; Kerisit 1987; Kerisit et al. 1987) and to *Magic Sets* (Bancilhon et al. 1986; Beeri and Ramakrishnan 1987b). In both methods, programs resulting from the corresponding algorithm may be unstratified. Thus, we introduce a new concept more general than stratification to capture the form of the transformed programs *(Weak Stratification)*, for which we define a particular model: the *W-Model.*

A related work to our has been recently proposed concerning the extension of Magic Sets to Stratified Databases (Balbin et al. 1987); but it does not guarantee that the query processing avoids redundancy in every case.

The organization of the paper is as follows. Section 1 provides notations and necessary background information concerning Logic Programs and Negation.

In section 2, we adapt the Alexander Method and Magic Sets to the treatment of Stratified Databases and point out that programs transformed by the revised methods may be unstratified. It leads, in section 3, to the characterization of a new class of logic programs so-called *Weakly Stratified*, for which we give a fixed-point semantics by means of their *W-Model*. Section 4 contains theoretical results concerning soundness and completeness of the method.

1. PRELIMINARIES

1.1 Notations

In the sequel, $X,Y,Z,X_i,...$ denote variables, $a,a_i,...$ denote constants, l,l_i,lh_i,lc denote literals, Greek letters denote substitutions. Given a substitution σ and an expression E (term,literal or clause), we denote by $E\sigma$ the application of σ on E. We denote by $c,c_i,...$ sign-operators: *identity* or *not*. If c is a sign-operator and l a literal, $c \lozenge l$ stands for l if c is *identity*, for *not* l if c is *not*.

We consider in the following a DDB (or a logic program) as a finite set of first order function-free clauses. A clause is a formula of the form: $lc \leftarrow l_1,..,l_n$ where $l_1,..,l_n$ are literals, $n \geq 0$, and lc is a positive literal. In definite clauses, each l_i must be positive; definite programs are logic programs made of definite clauses. More precisely, we only consider clauses with predefined variables: each variable which occurs in a negative l_i or in lc also appears in a positive l_j.

To distinguish transformed programs from original ones, we denote a clause $lc \leftarrow l_1,..,l_n$ in a transformed program by: $l_1,..,l_n \Rightarrow lc$ because these clauses are intended to be processed bottom-up (saturation).

Finally, we may write $l_1,..,l_n \Rightarrow lc_1,lc_2$ instead of:

$$\left[\begin{array}{l} l_1,..,l_n \Rightarrow lc_1 \\ l_1,..,l_n \Rightarrow lc_2. \end{array}\right.$$

for practical convenience.

1.2. Semantics of Logic Programs

The semantics of a logic program can be characterized by its Herbrand models (Lloyd 1984). A logic program can actually have a lot of models(1), but we are most interested in its minimal models. A model is the least model of a logic program P if it is the only minimal model of P. For a definite logic program, there exists such a least Herbrand model, and this model is generally intended to characterize its semantics (Lloyd 1984). Indeed, this model contains exactly all the facts which are true in every model of P.

Since only positive information can be inferred from a definite program, deducing negative information requires an additional inference mechanism. To state that a fact is supposed to be false if it can not be proved to be true is the most natural way to achieve it; this rule called *Closed World Assumption* (CWA) (Reiter 1978) is equivalent to say that a fact which is not in the least model is supposed to be false. Transposing it to the framework of SLD-Resolution led to the concept of *Negation as Failure*: a negative ground atomic goal *(← not B)* is said to be true if there is no SLD-derivation of the empty clause from *(← B)*. In fact, as SLD-derivations may be infinite, only *finite failure* can be checked. Thus, in the sequel, we denote by *SLD-NFF (Negation as Finite Failure)* the resolution method usually denoted by *SLD-NF*. Assuming that programs are function-free, deduction methods can check in finite time without loss of efficiency whether a fact belongs to the least model or not, (Bancilhon et al. 1986; Rohmer et al 1986; Vieille 1986). One can say that these methods implement CWA over function-free definite programs.

On the other hand, a program with negation may have several minimal Herbrand models. Thus the choice between the models needs supplementary knowledge. Then, if a particular model has been chosen, negative information can be derived (as for definite programs) by completion of the model towards the Herbrand base.

1.3. Stratification

As said before, if a logic program contains negation, it may have no least model.

Recently, two independent works (Apt et al. 1986; van Gelder 1986) led to the introduction of a new concept to overcome this difficulty: *Stratification*. The principle of this approach consists in disallowing recursion *through* negation.

(1) As the logic programs we consider don't contain function symbols, they only have a finite number of models and all of them are finite.

A logic program is said to be *stratified* if it can be divided into ordered strata S_i for i=1,...,n such that:

- If a relation symbol occurs positively in a clause in S_i, then its definition(2) is contained in $\cup S_j$ for $j \leq i$

- If a relation symbol occurs negatively in a clause in S_i, then its definition is contained in $\cup S_j$ for $j<i$

For a stratified program $P = \cup S_i$ (i=1,...,n), its semantics is well-defined by a model M which can be computed as follows:
$M_1 = T_{S1}\uparrow\omega(\emptyset) \dots\ M_n = T_{Sn}\uparrow\omega(M_{n-1})$; $M = M_n$
This model is called the *natural model* of P (Apt and Pugin 1987).

The motivation for this framework is the following: if the negation of an atom (say *not B*) is to be supposed (say for a rule in stratum S_i), one has to be sure that *B* cannot be proved in the sequel; it is the case here, since only rules in lower strata than S_i can produce positive information about *B*. For more details about stratification, see (Apt et al. 1986; van Gelder 1986; Apt and Pugin 1987).

2. DEDUCTION METHODS

We first recall how deduction methods apply for definite programs and transpose them intuitively for programs with negation; finally, we give a formal presentation of the Extended Alexander Algorithm.

2.1. Programs without Negation

Given a query to a DDB, a deduction method uses definitions from IDB to transform it into a combination of queries to EDB. Depending on the dynamic or static nature of the combination, classical methods are said to be *interpreted* (SLD-Resolution, APEX, QSQ, ...) or *compiled* (Alexander Method, Magic Sets, Counting, Magic Counting, ...).

Within an interpreted method, the evaluation strategy is described by a general algorithm specifying exactly the necessary computation steps to answer any query to the DDB (Demolombe and Royer 1986).

On the other hand, a compiled method transforms statically the DDB with respect to query modes into a new DDB. Then, the new DDB is to be processed using any bottom-up evaluation strategy. One interest of the compiled methods is that no control mechanism obscures the semantics of the evaluation and this makes proofs more easy and readable. Furthermore these methods can be easily coupled with any relational DBMS: bottom-up evaluation can be efficiently implemented using a mapping towards relational algebra (Rohmer et al. 1986; Ellul 1987). Finally, note that bottom-up evaluation allows a high level of parallelism: each rule can be executed on a distinct processor, each processor communicating with the others by sending tuples (Gonzalez et al. 1987).

(2) The definition of a predicate is the set of all clauses concluding on this predicate.

Thus we will focus in the sequel on two compiled methods: *Alexander Method* and *Magic Sets*. The principle of both methods is quite the same: it consists in simulating a top-down strategy using the definitions of IDB by producing a new program to be processed bottom-up over EDB.

Indeed, creating a new goal (e.g. on predicate p) will be simulated by the assertion of a new fact (*pb_p* in Alexander, *magic_p* in Magic Sets), and generating a new solution by the assertion of a corresponding new fact (*sol_p* in Alexander, *p* in Magic Sets).

Complete information about these methods can be found respectively in (Rohmer et al. 1986; Kerisit et al. 1987; Kerisit 1988) for Alexander and (Bancilhon et al. 1986; Beeri and Ramakrishnan 1987b) for Magic Sets.

Let us illustrate both methods on the same example (transitive closure).

Ex1:

$q(X,Y) \leftarrow p(X,Y)$ (1)
$q(X,Z) \leftarrow p(X,Y),q(Y,Z)$ (2)

Query: $\leftarrow q(a,X)$

We denote by *Q* the signature (instantiation mode) of the query (cf definition 2.3.1): in this example, *Q* stands for *q_1_0*.

Ex1.1: Alexander program

pb_Q(a). (1)
pb_Q(X),p(X,Y) ⇒ sol_Q(X,Y). (2)
pb_Q(X),p(X,Y) ⇒ pb_Q(Y),$cont_1$(X,Y). (3)
$cont_1$(X,Y),sol_Q(Y,Z) ⇒ sol_Q(X,Z). (4)

Ex1.2: Magic program with our notations

magic_Q(a). (1)
magic_Q(X),p(X,Y) ⇒ Q(X,Y). (2)
magic_Q(X),p(X,Y)⇒ supmagic_2_2(X,Y). (3)
supmagic_2_2(X,Y) ⇒ magic_Q(Y). (4)
supmagic_2_2(X,Y),Q(Y,Z) ⇒ Q(X,Z). (5)

2.2. Programs with Negation

Now, we consider DDBs where negation literals are allowed within the body of clauses. For such DDBs, the point is to propose a query answering system as powerful as previously.

Our approach is therefore to adapt the Alexander algorithm (Kerisit et al. 1987) and the Magic Sets algorithm (Beeri and Ramakrishnan 1987a) to the treatment of Stratified Databases. A negative goal (e.g *not p(X,Y)*) is transformed, as using the Negation as Failure principle, into a positive call (*pb_p* or *magic_p*) and a negative return (*not sol_p* or *not p*). Unfortunately, the programs produced by this method are not always stratified.

Example: a person who is not a Very Important Person but knows a Very Important Person must be jealous. A Very Important Person is rich or has rich ancestors.

Ex2:

IDB:

jealous(X) ← person(X), not vip(X), knows(X,Y),
  vip(Y). (1)
vip(X) ← rich(X). (2)
vip(Y) ← parent(X,Y),vip(X). (3)

(1) belongs to stratum S_2, (2),(3) belong to stratum S_1.

EDB: contains the 'person','parent','knows' and 'rich' relations in stratum S_0.

Ex2.1: Alexander transformed database for query "jealous(alex) ?"

IDB':

pb_j(X),person(X) ⇒ pb_v(X),$cont_1$(X). (1)
$cont_1$(X),not sol_v(X),knows(X,Y) ⇒
  pb_v(Y),$cont_2$(X,Y). (2)
$cont_2$(X,Y),sol_v(Y) ⇒ sol_j(X). (3)
pb_v(X),rich(X) ⇒ sol_v(X). (4)
pb_v(X),parent(Y,X) ⇒ pb_v(Y),$cont_3$(X,Y). (5)
$cont_3$(X,Y),sol_v(Y) ⇒ sol_v(X). (6)

where *j* and *v* stand for signatures of *jealous* and *vip*.

Rules (1), (2) & (3) belong to S'_2, the others to S'_1.

EDB': contains *EDB* and the fact *pb_j(alex)* in S'_0.

Ex2.2: Magic Transformed Database for query "jealous(alex) ?"

IDB':

magic_j(X),person(X) ⇒ supmagic_1_2(X). (1)
supmagic_1_2(X) ⇒ magic_v(X) (2)
supmagic_1_2(X),not v(X),knows(X,Y) ⇒
supmagic_1_5(X,Y). (3)
supmagic_1_5(X,Y) ⇒ magic_v(Y). (4)
supmagic_1_5(X,Y),v(Y) ⇒ j(X). (5)
magic_v(X),rich(X) ⇒ v(X). (6)
magic_v(X),parent(Y,X) ⇒ supmagic_3_2(X,Y). (7)
supmagic_3_2(X,Y) ⇒ magic_v(Y). (8)
supmagic_3_2(X,Y),v(Y) ⇒ v(X). (9)

Rules (1),(2),(3),(4) & (5) are in S'_2, the others in S'_1.

EDB': contains *EDB* and the fact *magic_j(alex)* in S'_0.

Let us present below a formal definition of the extension of the Alexander Method to Stratified Databases.

2.3. The Extended Alexander Algorithm

Before presenting the Extended Alexander Algorithm (EAA), we give a formal definition of signatures.

2.3.1. The Notion of Signature

Each signature S of a given predicate p (with arity n) has to capture one instantiation mode for p. A signature has four components:
- *a name* S = p_a_1_..._a_n where each a_i describes the instantiation mode of the i^{th} argument: a_i is 1 if the i^{th} argument is bound, 0 else;
- *a projection operator* πS which associates to any n_tuple t a tuple containing only those arguments of t which are said to be bound in S;
- a *problem* predicate pb_S with arity equal to the number of bound arguments in S;
- a *solution* predicate sol_S with arity equal to n.

Now, a *Sideways Information Passing (SIP) strategy* (Beeri and Ramakrishnan 1987b) determines for a given clause and a given signature of its head the derived signatures of the tail's predicates. In the sequel, we don't define particular SIP-strategies. However, in order to avoid unsafe computation, we do not allow SIP-strategies where the induced signatures of negative literals contain a *0* symbol: i.e. a negative literal can be a goal iff it is fully instantiated.

2.3.2. The detailed Algorithm (EAA)

Given an atomic query q to a stratified DDB B, EAA produces a new DDB $B'=(EDB',IDB')$.

CAND denotes in the following the set of couples (C,S), C being a candidate clause with associated signature S;

OLD denotes the set of couples in *CAND* already treated.

```
EAA:    input B = EDB ∪ IDB
        input q
        output B' = EDB' ∪ IDB'
BEGIN

INITIALIZATION :
We associate with q a signature S in which constants are
represented as input arguments and variables as output
arguments.
EDB' = { pb_S(constants) } ∪ EDB
CAND = { (C,S) | head of clause C has the same
      relation name as q }
OLD = ∅

While (CAND ≠ OLD)
{   Choose a pair (C,S) from (CAND - OLD);
    let s be the stratum number of C;

    Add (C,S) to OLD

    If C is a unit clause                  ( C : lc(t) ← )
    Add pb_S(πS.t) ⇒ sol_S(t) to IDB' in stratum S'_s

    Else          ( C : lc(t) ← lh_1(t_1), ... , lh_n(t_n). )
    { Add the following rules to IDB' in stratum S'_s:

        pb_S(πS.t) ⇒ pb_S_1(πS_1.t_1),cont_1(t'_1).

        cont_1(t'_1), c_1 ◊ sol_S_1(t_1) ⇒
        pb_S_2(πS_2.t_2),cont_2(t'_2).
                 . . .
        cont_n(t'_n), c_n ◊ sol_S_n(t_n) ⇒ sol_S(t).

        Each clause concluding on a predicate lh_i
        becomes a candidate with signature S_i.
    } END Else
} END While
END
```

Notations:

* S_i are signatures associated with lh_i as explained above.

* $cont_i$ are new predicates.

* for each i, t'_i is the tuple containing those variables which occur in both of the expressions:

$pb\_S(\pi S.t),\ldots,lh_{i-1}(t)$ and $lh_i(t_i),\ldots,lh_n(t_n) \Rightarrow sol\_S(t)$.

* c_i is the sign-operator corresponding to lh_i (*not* if lh_i is negative, *identity* else); for any positive literal l, '*not* ◊ l' stands for '*not* l' and '*identity* ◊ l' for 'l'.

2.4. Unstratification

One can easily notice that neither the Alexander transformed program in the former example (Ex2.1) nor the Magic transformed program (Ex2.2) are stratified since there exists a cycle containing a negation: (pb_v → sol_v → pb_v) and (magic_v → v → supmagic_1_5 → magic_v). Indeed, unstratification in this example is due to the presence of both negative and positive occurrences of the same predicate (*vip*) in the same clause (clause (1) in Ex2).

When dealing with a stratified transformed program, the semantics of the natural model seems to guarantee soundness and completeness of the query answering. Anyway, we prove in section 4 a more general result.

The original feature of our work consists in showing that even unstratified transformed programs can be given a semantics which guarantees soundness and completeness. To do so, we prove that the transformed programs belong to a new class of programs we call 'Weakly Stratified'.

3. WEAK STRATIFICATION

3.1. Definition

A program P is said to be *weakly stratified* iff there is a partition $P = \cup S_i$ (i=1,...,n) such that:

for each i, if a relation symbol occurs negatively in a clause in stratum S_i, then its definition is contained in $\cup S_j$ $(j<i)$.

It means that only the first syntactical condition of stratification is retained. As a matter of fact, a stratified program is weakly stratified.

Ex3: Example of a weakly stratified program which is not stratified.

p(X) ⇒ q(X)
r(X),not q(X) ⇒ p(X)

Ex4: Example of a program which is not weakly stratified.

$r(X,Y)$, not $p(X) \Rightarrow p(Y)$

In the general case, it is difficult to characterize a natural semantics for such a program. The semantics we adopt here consist in using a rule which contains a negation on a given predicate (say *p*) only when no rule of a lower stratum can produce any more positive information for this predicate *p*. We shall define below a fixed-point operator which determines the choice of a model, called the *W-model*. It will be shown that, in the case of a stratified program, the W-model coincides with the natural model defined by Apt et al. (1986).

3.2. Definition of Tw

Let P be a weakly stratified program, divided into strata S_i ($i=1,\ldots,n_s$). For each i, let N_i be the set of clauses in S_i having a negative literal in their body. Let $Neg = \cup N_i$ ($i=1,\ldots,n_s$) and Pos the set of clauses without negation. Let U be the Herbrand Base associated with P.
We define Tw_P the fixed-point operator associated with P as a mapping from P(U) to P(U) such that:

$$Tw_P(I) = \begin{cases} I \cup T_{Pos}(I) & \text{if } T_{Pos}(I) - I \neq \varnothing \\ \text{else} \quad I \cup T_{N1}(I) & \text{if } T_{N1}(I) - I \neq \varnothing \\ \text{else} \quad \ldots & \\ \text{else} \quad I \cup T_{Nns}(I) & \end{cases}$$

where *T* is the immediate consequence operator extended to general programs (Apt et al. 1986):
$a \in T_P(I)$ iff there is a ground instance of a clause in *P* concluding on *a* with each positive atom of its body in *I* and no negated atom of its body in *I*.

We first give the following result concerning Tw:

Proposition 1:

M' is a Herbrand model of a weakly stratified program P iff it is a fixed-point of Tw_P

Proof is immediate.

The point is now to choose a particular model, which corresponds to the intuitive semantics.

3.3 Characterization of the W-Model

Definition:

We define M' the W-model of P as the minimal set of "safe" information obtained by iteration of Tw_P on the empty set. It follows that M' has to be the limit, if one exists, of $Tw_P\uparrow\alpha(\varnothing)$.

We prove the following properties for this model:

Proposition 2:

There exists an integer α such that :
$\forall\beta \geq \alpha \quad Tw_P\uparrow\beta(\varnothing) = Tw_P\uparrow\alpha(\varnothing) = M'$

Proof:
$Tw_P\uparrow\alpha(\varnothing)$ is an increasing sequence bounded by the finite Herbrand Base.

Proposition 3:

If P is a stratified program, the W-model of P is also the natural model of P

Sketch of proof:

If P is stratified, the construction of the W-model of P corresponds quite exactly to the construction of its natural model. The only difference is that positive rules may be used in a different order. Since using positive rules guarantees monotony, one can easily prove that the W-model of P and its natural model coincide. The complete proof can be found in (Kerisit 1988).

4. THEORETICAL RESULTS

The choice of a partition for transformed programs is as follows: we associate to a rule the same stratum number as the one of the original clause from which it was produced.

This partitioning policy leads to the following result:

Theorem 1:
Transformed programs are weakly stratified.

Proof is immediate using the stratification condition of the original program.

The major result of our work is the proof of completeness and soundness of EAA. Actually, soundness and completeness are to be proved simultaneously: suppose for instance that a ground atom cannot be proved because of incompleteness, then any clause using the negation of this atom may produce incorrect deduction.

Theorem 2:

Given B a stratified DDB, q(t) a query to B (with signature S), let B' be the DDB obtained by Alexander transformation of B with q(t). The answers q(u) to the query correspond exactly with the sol_S(u) which belong to the W_model of B'.

Sketch of proof:

Intuitively, proof is same as for Horn clauses programs excepted for negative information (only solution-predicates may be negated in the transformed programs). Any transformed rule (say *R*) containing a negated solution predicate (say *sol_S*) has been produced together with a rule defining its corresponding problem predicate (*pb_S*). Thus, according to Tw's strategy, only the ground instances of *not sol_S* corresponding to already proved *pb_S* instances may be used when rule *R* is invoked. On the other hand, the fixed-point operator ensures that all rules able to prove these *sol_S* instances (directly or not) have been used before rule *R* is used with these instances. The complete proof, made by induction on the strata, can be found in (Kerisit 1988).

CONCLUSION

Our work proposes a way to design efficient query evaluation systems for stratified databases. To achieve it, we extend the Alexander algorithm and present a new bottom-up execution strategy dedicated to the new class of so-called 'weakly stratified' programs. This method can be seen as an efficient implementation of Negation as Failure (not finite failure) for stratified programs.

Moreover, comparison with related work (Balbin et al. 1987) seems to favour our method for its simplicity and efficiency. In this paper, the authors solve the problem of unstratification of the transformed programs by adding a preliminary technique to the extended Magic Algorithm called *BPR labelling*. However, this technique involves a lot of duplications which in turn leads to some inefficiency. Our approach preserve the simplicity of the original methods and improve efficiency with a new bottom-up evaluation strategy.

ACKNOWLEDGEMENTS

We would like to thank all members of the A.I. Dept. and especially Jean Rohmer & Rémi Lescoeur for their support and their encouragements.

REFERENCES

(Apt et al.1986)
K.R.Apt, H.Blair, A.Walker: "Towards a Theory of Declarative Knowledge", Tech. Report 86-10, IBM Research Center of Yorktown.

(Apt and Pugin 1987)
K.R.Apt, J.M.Pugin: "Maintenance of Stratified Databases viewed as a Belief Revision System", Proc. of 6th ACM symp. on Princ. of Database Systems , San Diego March 1987.

(Bancilhon and Ramakrishnan 1986)
F.Bancilhon, R.Ramakrishnan: "An amateur's introduction to recursive query processing strategies", Proc. of the ACM-SIGMOD conf. Washington DC, May 1986.

(Beeri and Ramakrishnan 1987a)
C.Beeri, R.Ramakrishnan: "On the Power of Magic", The Hebrew University Draft Jan. 1987.

(Beeri and Ramakrishnan 1987b)
C.Beeri, R.Ramakrishnan: "On the Power of Magic", Proc. of 6th ACM symp. on Princ. of Database Systems , San Diego March 1987.

(Bancilhon et al. 1986)
F.Bancilhon, D.Maier, Y.Sagiv and J.D.Ullman: "Magic Sets and other strange ways to implement Logic Programs", Proc. of the ACM SIGACT-SIGMOD Symp. on Princ. of Database Systems 1986.

(Balbin et al. 1987)
I.Balbin, G.S.Port, K.Ramamohanarao: "Magic Set Computation for Stratified Databases", Tech. Report of Univ. of Melbourne, 1987.

(Clark 1978)
K.L.Clark: "Negation as Failure" in "Logic and Databases", Plenum Press, New York 1978.

(Demolombe and Royer 1986)
R.Demolombe, V.Royer (ONERA-CERT): "Evaluation Strategies for Recursive Axioms: a Uniform Representation", prel. draft, October 1986.

(Ellul 1987)
A.Ellul: "Couplage entre mécanisme de Déduction et Base de Données", proc. of "Des Bases de Données aux Bases de Connaissances" AFCET conference, Sophia Antipolis, France, Sept. 1987.

(Gardarin and de Maindreville 1986)
G.Gardarin, C.De Maindreville: "Evaluation of Database Recursive Programs as Recursive Function Series", Proc. ACM-SIGMOD Int. Conf. on Management of Data, Washington DC, May 1986.

(Gallaire and Minker 1978)
H.Gallaire, J.Minker (eds.) : "Logic and Databases", Plenum, New York 1978.

(Gonzalez et al. 1987)
R.Gonzalez-Rubio, J.Rohmer, A.Bradier: "An Overview of DDC: Delta Driven Computer", BULL Research Center, Technical Report DMIA 87007 March 87.

(Kerisit 1987)
J.M.Kerisit: "A Relational Approach to Logic Programming: the Extended Alexander Method", BULL Research Center, Technical Report DLA/SLIA 87004 Febr. 1987.

(Kerisit 1988)
J.M.Kerisit: "La méthode d'Alexandre: une technique de Déduction", Thèse de Doctorat, Université Paris VII, June 1988.

(Kifer and Lozinskii 1986a)
M.Kifer, E.L.Lozinskii: "Filtering Data Flow in Deductive Databases", Proc. of the Int.Conf. on Database Theory, Rome, Italy, Sept. 1986.

(Kifer and Lozinskii 1986b)
M.Kifer, E.L.Lozinskii: "Can We Implement Logic as a Database System ?", Techn. Report 86/6 of the State University of New-York at Stony Brook, March 1986.

(Kerisit et al. 1987)
J.M.Kerisit, R.Lescoeur, J.Rohmer, G.Roucairol: "The Alexander Method: an efficient way for handling Deduction on Data Bases", BULL Research Center, Technical Report DLA/SLIA 87015 June 1987.
Also in "Programming Future Generation Computers", pp 283-304, K.Fuchi & M.Nivat ed., North Holland, 1988.

(Lloyd 1984)
J.W.Lloyd: "Foundations of Logic Programming", Springer Verlag 1984.

(Pugin 1988)
J.M.Pugin: "Contribution à l'Etude des Raisonnements Non Monotones: le Tableur Logique", Thèse de Doctorat, Université Paris VII, June 1988.

(Rohmer et al. 1986)
J.Rohmer, R.Lescoeur, J.M.Kerisit: "The Alexander method, A technique for the processing of recursive Axioms in Deductive Databases", New Generation Computing 1986-4(3): p.273..285.

(Sacca and Zaniolo 1986)
D.Sacca, C.Zaniolo : "On the implementation of a Simple Class of Logic Queries for Databases", Proc. of the 5th ACM SIGACT-SIGMOD Symp. on Princ. of Database Systems, Cambridge MA, March 1986.

(Ullman 1985)
J.D.Ullman: "Implementation of Logical Query Languages for Databases", ACM Trans. on Database System 10(3) p 289..321 Sept. 1985.

(van Gelder 1986)
A.Van Gelder: "Negation as failure using Tight Derivations for General Logic Programs", Proc. third IEEE Sympos. on Logic Programming, Salt Lake City 1986.

(Vieille 1986)
L.Vieille: "Recursive Axioms in Deductive Databases: the Query/SubQuery Approach", proc. of the first Int. Conf. on Expert Database Systems. Charleston April 1986.