

A Uniform Framework for Deductive Database Derivation Strategies

Robert DEMOLOMBE
ONERA/CERT
2 av. Edouard Belin, B.P. 4025
31055 Toulouse,
FRANCE
demolomb@tls-cs.cert.fr

February 4, 2004

Abstract

A uniform framework is presented to describe the most typical strategies that are used to compute answers to Deductive Databases. The framework is based on the definition of a general Least Fixpoint operator that operates on meta rules. Each set of meta rules represents a different strategy, and this allows an easy comparison.

We first consider Deductive Databases with Horn clauses and we present strategies based on the general ideas of Forward chaining, Backward chaining, and their combinations. We show that well known strategies, like Magic Sets, can be represented in this general framework, and can be compared to other ones.

Then we consider non Horn clauses, and we present strategies to compute Intensional answers and Conditional answers.

The purpose of the paper is mainly didactic even if new strategies are presented in the last section. The presented framework should make easier comparisons between existing strategies.

Keywords: Deductive Databases, Automated Deduction

1 Introduction

Deduction strategies to compute answers to queries to Deductive Databases combine ideas that come from query optimization in Relational Databases, and from Prolog

(SLD resolution). In the last few years there was a lot of work on the definition of Deductive Database strategies [3, 10, 18, 23, 24, 28, 27, 14] [19, 11, 15, 17, 25, 26, 2], and it is difficult to have a global view of the state of the art because authors present these strategies in many different ways, and sometimes it is not easy to realize that several strategies are in fact based on the same intuitive ideas.

It is our claim that most of them are based on a very limited number of fundamental ideas like: Backward chaining, Forward chaining, and unification process to select relevant facts for a given query.

The objective of this paper is to present a uniform framework to represent these strategies, and to present the most typical ones in this framework. We believe that this presentation can also be fruitful to design new strategies. A similar attempt was presented in [9] in a much more informal way. The paper is written in a semi-formal style in order to avoid technical details that could hide the main ideas, and because there are some open theoretical questions. However it is formal enough to define formal proofs for their properties.

In section 2 is presented the general framework. It is based on the definition of a Least Fixpoint operator that operates on meta rules. A given set of meta rules defines a particular strategy, and, to compare different strategies we just have to compare these meta rules. The idea of using meta rules to define “à la Prolog” strategies is not really new and was used in [4] to present the Magic Sets strategy. It will be shown that the presented framework is much more general, in particular because it allows to consider answers as sets of clauses and not just as sets of substitutions.

In section 3 are presented strategies to compute answers in the context of Horn clauses. We first remind the simplest strategies, and then we show how more sophisticated ones, presented in the literature, can be represented in this framework.

Finally in section 4 we present strategies to compute answers in the context of non Horn clauses, in order to provide new kinds of answers: intensional answers and conditional answers. They are relatively new, and this shows how the general framework allows to extend the field of Deductive Databases to new applications.

2 Least Fixpoint Framework

In this section are given the definitions of a Deductive Database, of the associated meta level language for strategy descriptions, and of the Least Fixpoint operator that computes answers according to a given strategy.

Definition: Deductive Database DB

IDB : set of Range Restricted clauses. A clause is Range Restricted iff each variable that appears in a positive literal also appears in a negative literal.

EDB : set of ground clauses.

DB = IDB \cup EDB

Definition: Language L_{DB} of a Deductive Database DB

The language L_{DB} of DB is a First Order language, whose predicates are the predicates that appear in DB, whose constants are the constants that appear in DB, and whose variables is an infinitely enumerable set of variables.

Definition: Language L_{MDB} of a Meta Database MDB of DB

The language L_{MDB} is defined in function of the language L_{DB} . The language is a Many Sorted First Order language, and the predicates of L_{MDB} are the following “meta predicates”:

- $Q(x)$: x is a literal of L_{DB} that represents a query.
- $Ax(x)$: x is a clause of L_{DB} that appears in DB.
- $Th(x)$: x is a clause of L_{DB} which is a theorem derivable from DB.

The constants of L_{MDB} are the literals of L_{DB} . These literals may contain variables of L_{DB} .

There are two function symbols in L_{MDB} : $(x) \vee (y)$, and $\neg(x)$.

Terms of L_{MDB} are syntactically restricted to terms where arguments of $\neg(x)$ are variables or constants of L_{MDB} .

Variables of L_{MDB} are of one of the following sorts:

- variables of sort “literal of L_{DB} ”, denoted by: l, l_1, l_2, \dots . Their sort is the set of literals of L_{DB} .

- variables of sort “literal or disjunction of literals of L_{DB} ”, denoted by: c, c_1, c_2, \dots . Their sort is the set of literals or clauses of L_{DB} .

- variables of sort “literal or conjunction of literals of L_{DB} ”, denoted by: h, h_1, h_2, \dots . Their sort is the set of literals of conjunction of literals of L_{DB} .

Notations: We adopt the following notations:

1. Terms are denoted by: $t_1 \vee t_2 \vee \dots \vee t_n$, or may be denoted as well by: $t_1 \vee \dots \vee t_i \leftarrow t_{i+1} \wedge \dots \wedge t_n$, where the t_i s do not contain disjunction symbol, and

where \tilde{t}_j denotes s_j , if t_j denotes $\neg s_j$, and \tilde{t}_j denotes $\neg s_j$, if t_j denotes s_j , and s_j does not contain negation symbol.

2. It is assumed that: $t_1 \vee t_2 \vee \dots \vee t_n$, and: $t'_1 \vee t'_2 \vee \dots \vee t'_n$ denote the same term iff $\{t_1, t_2, \dots, t_n\} = \{t'_1, t'_2, \dots, t'_n\}$.
3. If $M(t_1 \vee t_2 \vee \dots \vee t_n)$, and $M(t'_1 \vee t'_2 \vee \dots \vee t'_n)$ are ground atoms of L_{MDB} , it is assumed that they denote the same ground atom iff $t_1 \vee t_2 \vee \dots \vee t_n$ is a variant, in L_{DB} , of $t'_1 \vee t'_2 \vee \dots \vee t'_n$.

Definition: Object Instance of a Ground Atom

Let mf' be a ground atom of L_{MDB} of the form: $M(L'_1, L'_2, \dots, L'_n)$, where M is a predicate of L_{MDB} , and the L'_i 's are the literals of L_{DB} that appear in mf' .

Let x_1, x_2, \dots, x_p be the variables of L_{DB} that appear in some L'_i .

Let s be a substitution of terms of L_{DB} of the form $s = \{t_1/x_1, \dots, t_p/x_p\}$.

We say that mf is an object instance of mf' iff we have: $mf = mf'.s$.

Definition: Object Instance of a Set of Ground Atoms

Let $F = \{mf_1, mf_2, \dots, mf_n\}$ and $F' = \{mf'_1, mf'_2, \dots, mf'_p\}$ be two sets of ground atoms of L_{MDB} , we say that F is an object instance of F' iff each mf_i in F is an object instance of some mf'_j in F' .

Definition: Set of Ground Atoms Satisfying a Rule

Let F and F' be two sets of ground atoms of L_{MDB} of the form: $F = \{mf_1, mf_2, \dots, mf_n\}$, and $F' = \{mf'_1, mf'_2, \dots, mf'_p\}$, such that F is an object instance of F' .

Let gir be a ground clause of L_{MDB} of the form: $mf_1 \wedge mf_2 \wedge \dots \wedge mf_n \rightarrow mf$.

We say that the object instance F of F' satisfies the rule r iff F is a most general object instance of F' such that gir is a ground instance of r .

Example:

Let r be the rule: $Q(l) \wedge Ax(l \leftarrow h) \rightarrow Th(l \leftarrow h)$.

Let F_1 and F' be the two sets of ground atoms:

$$F' = \{Q(p(a, x)), Ax(p(x, y) \vee p(y, x) \vee \neg q(x, y))\}$$

$$F_1 = \{Q(p(a, x)), Ax(p(a, x) \leftarrow q(a, x) \wedge \neg p(x, a))\}$$

The object instance F_1 of F' satisfies r because

$Q(p(a, x)) \wedge Ax(p(a, x) \leftarrow q(a, x) \wedge \neg p(x, a)) \rightarrow Th(p(a, x) \leftarrow q(a, x) \wedge \neg p(x, a))$ is a ground instance of r , and there is no more general instance of F' that has the same property.

Let F_2 be the set of ground atoms:

$$F_2 = \{Q(p(a, x)), Ax(p(a, x) \leftarrow q(x, a) \wedge \neg p(x, a))\}$$

If we consider the ground instance of r:

$$Q(p(a, x)) \wedge Ax(p(a, x) \leftarrow q(x, a) \wedge \neg p(x, a)) \rightarrow Th(p(a, x) \leftarrow q(x, a) \wedge \neg p(x, a))$$

we can see that the object instance F_2 of F' also satisfies r.

This example also shows that a given atom in a rule, like $Ax(l \leftarrow h)$, due to notation conventions, can be unified into several different ways to a given ground atom. Here $Ax(p(x, y) \leftarrow q(x, y) \wedge \neg p(y, x))$, and $Ax(p(y, x) \leftarrow q(x, y) \wedge \neg p(x, y))$, are two different representations of the same atom that lead to the unifications: $\{l=p(x, y), h=q(x, y) \wedge \neg p(y, x)\}$, or $\{l=p(y, x), h=q(x, y) \wedge \neg p(x, y)\}$.

Definition: Meta Database MDB

$$MEDB = \{ Ax(c) : c \text{ is in } DB \} \cup \{ Q(L) \}, \text{ where } L \text{ is a literal of } L_{DB}.$$

$$MIDB = \text{set of Range Restricted Horn clauses of } L_{MDB} \text{ without constants.}$$

$$MDB = MEDB \cup MIDB$$

Example:

$$IDB = \{ p(x) \leftarrow q(x) \wedge r(x), q(x) \leftarrow s(x) \}$$

$$EDB = \{ q(a), r(a), s(b), r(b) \}$$

$$MEDB = \{ Ax(p(x) \leftarrow q(x) \wedge r(x)), Ax(q(x) \leftarrow s(x)), Ax(q(a)), Ax(r(a)), Ax(s(b)), Ax(r(b)), Q(p(b)) \}$$

$$MIDB = \{ Ax(l) \rightarrow Th(l), Ax(l \leftarrow l_1 \wedge \dots \wedge l_n) \wedge Th(l_1) \wedge \dots \wedge Th(l_n) \rightarrow Th(l) \}$$

Definition: Herbrand Universe of MDB

The Herbrand Universe U_{MDB} of MDB is the set of ground terms of the language L_{MDB} , i.e. the set of clauses of the language L_{DB} .

Definition: Herbrand Base of MDB

The Herbrand Base H_{MDB} of MDB is the set of ground atoms formed with elements of U_{MDB} .

Definition: Closure Operator T_{MDB}

The definition domain of T_{MDB} is $: 2^{H_{MDB}} \rightarrow 2^{H_{MDB}}$.

Let S and S' be two subsets of $2^{H_{MDB}}$. We have $S' = T_{MDB}(S)$ iff

$$S' = \{ mf : mf \in S \text{ or } mf \in MEDB \text{ or} \\ \text{there exists a ground instance of a rule } r \text{ in } MIDB: \\ mf_1 \wedge mf_2 \wedge \dots \wedge mf_n \rightarrow mf \\ \text{such that: } \{mf_1, mf_2, \dots, mf_n\} \text{ is an object instance of } \{mf'_1, mf'_2, \dots, mf'_n\} \\ \text{that satisfies } r, \text{ and } \{mf'_1, mf'_2, \dots, mf'_n\} \text{ is included in } S \}$$

Property1: Complete Lattice

If we consider the set inclusion as an ordering relation over $2^{\text{H}_{\text{MDB}}}$, then $2^{\text{H}_{\text{MDB}}}$ is a complete lattice.

Property2: Continuity of T_{MDB}

The operator T_{MDB} is continuous.

See the proof in Appendix.

Property3: Least Fixpoint of T_{MDB}

The operator T_{MDB} has a least fixpoint, and this least fixpoint is T_{MDB}^ω .

This is a direct consequence of Property2.

Definition: Incremental Closure Operator I_{MDB}

The definition domain of I_{MDB} is $: 2^{\text{H}_{\text{MDB}}} \times 2^{\text{H}_{\text{MDB}}} \rightarrow 2^{\text{H}_{\text{MDB}}} \times 2^{\text{H}_{\text{MDB}}}$.

Let $S, S', D,$ and D' be subsets of $2^{\text{H}_{\text{MDB}}}$. We have $(S', D') = I_{\text{MDB}}(S, D)$ iff

- if $(S, D) = (\emptyset, \emptyset)$ then $(S', D') = (\text{MEDB}, \text{MEDB})$

- else, let N be defined by:

$$N = \{ \text{mf} : \text{there exists a ground instance of a rule } r \text{ in MIDB:} \\ \text{mf}_1 \wedge \text{mf}_2 \wedge \dots \wedge \text{mf}_n \rightarrow \text{mf} \\ \text{such that: } \{ \text{mf}_1, \text{mf}_2, \dots, \text{mf}_n \} \text{ is an object instance of } \{ \text{mf}'_1, \text{mf}'_2, \dots, \text{mf}'_n \} \\ \text{that satisfies } r, \text{ and } \{ \text{mf}'_1, \text{mf}'_2, \dots, \text{mf}'_n \} \text{ is included in } S, \\ \text{and for some } i \text{ in } [1, n], \text{mf}'_i \text{ is in } D \}$$

we have: $S' = S \cup D$, and $D' = S' - S$.

Property4: Correspondance between the Operators I_{MDB} and T_{MDB}

Let S_n be defined by: $S_n = T_{\text{MDB}}^n(\emptyset)$, then we have: $(S_n, S_n - S_{n-1}) = I_{\text{MDB}}^n(\emptyset, \emptyset)$.

See the proof in Appendix.

Property4 allows to compute $T_{\text{MDB}}^\omega(\emptyset)$ by computing $I_{\text{MDB}}^\omega(\emptyset, \emptyset)$. The operator I_{MDB} is much more efficient than T_{MDB} in the sense that it computes much less consequences. In the rest of the paper it will be assumed that the consequences of a given MDB are computed with the operator I_{MDB} .

3 Horn Clause Deductive Databases

In this section it is assumed that clauses in IDB are Horn clauses of L_{DB} , and that clauses in EDB are ground atoms of L_{DB} .

3.1 Forward Chaining

This strategy has still been presented in a previews example. It is presented here mainly for exhaustivity. It does not correspond exactly to what is called Forward chaining in the field of Automated Deduction [6], because not all the consequences are generated. Only consequences corresponding to ground atoms of L_{DB} are generated.

This strategy is described by the following MIDB:

- (1) $Ax(l) \rightarrow Th(l)$
- (2) $Ax(l \leftarrow l_1 \wedge l_2 \wedge \dots \wedge l_n) \wedge Th(l_1) \wedge Th(l_2) \wedge \dots \wedge Th(l_n) \rightarrow Th(l)$

The variable l is of type “literal of L_{DB} ”, then, in $Ax(l)$, l can only be unified with some L which is a ground atom of EDB .

If in $Th(l_1), \dots, Th(l_n)$, the variables l_1, \dots, l_n are unified to ground atoms of EDB , all the variables in the atom of L_{DB} unified with l are instantiated by constants of L_{DB} , because the clauses in IDB are Range Restricted, and all the variables in the positive literal unified to l appear in some literal unified to some l_i . Therefore in the generated consequence $Th(l)$, l is unified to a ground atom of L_{DB} .

From this two facts, it can be easily proved, by induction, that all the consequences computed by I_{MDB} are of the form $Th(L)$, where L is a ground atom of L_{DB} . Since there is no function symbol in L_{DB} , the number of distinct ground atoms of L_{DB} is finite, and the number of generated $Th(L)$ is also finite. For this reason the process stops after a finite number of steps.

Features:

- Valid and Complete (with respect to the derivation of ground atomic consequences of DB).
- Efficiency: very poor.
- Termination: yes.

3.2 Backward Chaining

This strategy is described by the following MIDB:

- (1) $Q(l) \wedge Ax(l \leftarrow l_1 \wedge l_2 \wedge \dots \wedge l_n) \rightarrow Th(l \leftarrow l_1 \wedge l_2 \wedge \dots \wedge l_n)$
- (2) $Q(l) \wedge Th(l \leftarrow l_1 \wedge \dots \wedge l_i \wedge \dots \wedge l_n) \wedge Ax(l_i \leftarrow h_i) \rightarrow Th(l \leftarrow l_1 \wedge \dots \wedge h_i \wedge \dots \wedge l_n)$
- (3) $Q(l) \wedge Th(l \leftarrow l_1 \wedge l_2 \wedge \dots \wedge l_n) \wedge Ax(l_1) \wedge Ax(l_2) \wedge \dots \wedge Ax(l_n) \rightarrow Th(l)$

The first idea in the forward chaining strategy is to select rules in IDB that allow

to derive consequences that unify with the query. That is represented by the rule (1).

Then, if some rule in IDB allows to derive consequences that are antecedents of a rule that derives answers to the query, the body of the former rule is substituted to the corresponding antecedents. That is represented by the rule (2).

This process can be viewed as “unfolding”, in the Logic Programming terminology. In Automated Deduction terminology it corresponds to an Input Resolution strategy, where the Input set is DB, with the additional constraint that **each resolvent must contain an instance of the query l**.

The rule (3) shows that answers to the query l are generated only when each l_i can be unified with an axiom in EDB.

Here the computation may never stop, because there are generated consequences of the form: $\text{Th}(l \leftarrow h)$, where: $l \leftarrow h$ is not necessarily a ground clause of L_{DB} , and it is possible to derive consequences where the number of l_i s is not bounded.

Let’s consider, for instance, the very well known example of the “ancestors”. If we have in MEDB:

$$\text{Ax}(\text{anc}(x, y) \leftarrow \text{parent}(x, z) \wedge \text{anc}(z, y)), \text{ and } \text{Q}(\text{anc}(a, y))$$

the computation derives an infinite number of consequences of the form:

$$\text{Th}(\text{anc}(a, y) \leftarrow \text{parent}(a, z_1) \wedge \text{parent}(z_1, z_2) \wedge \dots \wedge \text{parent}(z_{n-1}, z_n) \wedge \text{anc}(z_n, y))$$

However it is not always the case that when IDB contains recursive definitions the computation generates an infinite number of consequences. Indeed, for a given query, it may happen that the computation involves a subset of IDB that contains no recursive definition. This can be checked using an MIDB that only contains the rule: ¹

$$\text{Q}(l) \wedge \text{Ax}(l \leftarrow l_1 \wedge \dots \wedge l_i \wedge \dots \wedge l_n) \rightarrow \text{Q}(l_i)$$

and removing from MEDB all the axioms that come from EDB. If MEDB contains $\text{Q}(L)$, the computation of I_{MDB} , with this unique rule, will generate the set of all “sub-queries” corresponding to $\text{Q}(L)$. Notice that this computation finitely stops, because the number of distinct atoms in L_{DB} is finite, provided that two variants are considered as the same atom.

If in this computation are never generated two consequences $\text{Q}(L_1)$ and $\text{Q}(L_2)$ such that L_2 is a variant of L_1 , then we are guaranteed that the computation of I_{MDB} , with the complete MEDB, will stop for the query $\text{Q}(L)$. In fact the rule in MIDB defines a method to traverse the connection graph of the set of clauses that

¹To make clear the meaning of this rule, we have to say that as many $\text{Q}(l_i)$ can be derived using this rule as they are literals l_1, l_2, \dots, l_n .

are in IDB. If the same sub-query is generated twice, that means that there is a path, starting from a clause which contains a literal unifiable with L, and which contains a cycle. This method provides a sufficient condition, of course it is not a necessary condition.

In the case where the Backward chaining strategy stops, it is certainly the most efficient one. The reason can be understood with the following simple example:

$$\text{MEDB} = \{ Q(p(x)), \text{Ax}(p(x) \leftarrow q(x)), \text{Ax}(q(x) \leftarrow r(x)), \text{Ax}(r(x) \leftarrow s(x)), \text{Ax}(s(a_1)), \dots, \text{Ax}(s(a_n)) \}$$

The computation of I_{MDB} generates:

1. $\text{Th}(p(x) \leftarrow q(x))$
2. $\text{Th}(p(x) \leftarrow r(x))$
3. $\text{Th}(p(x) \leftarrow s(x))$
4. $\text{Th}(s(a_1)), \dots, \text{Th}(s(a_n))$

No useless consequence of the form $\text{Th}(r(a_i))$ nor $\text{Th}(q(a_i))$ are generated.

Features:

- Valid and Complete (with respect to the derivation of ground atomic consequences of DB).
- Efficiency: excellent, when the computation stops.
- Termination: not necessarily; there is a general sufficient condition to check if the computation may stop.

3.3 Parallel Backward-Forward Chaining

The Backward-Forward chaining strategy tries to combine advantages of the two previous strategies, that is: termination, and no blind derivation of consequences.

This strategy is described by the following MIDB:

- (1) $Q(l) \wedge \text{Ax}(l \leftarrow l_1 \wedge \dots \wedge l_i \wedge \dots \wedge l_n) \rightarrow Q(l_i)$
- (2) $Q(l) \wedge \text{Ax}(l) \rightarrow \text{Th}(l)$
- (3) $Q(l) \wedge \text{Ax}(l \leftarrow l_1 \wedge \dots \wedge l_2 \wedge \dots \wedge l_n) \wedge \text{Th}(l_1) \wedge \text{Th}(l_2) \wedge \dots \wedge \text{Th}(l_n) \rightarrow \text{Th}(l)$

As mentioned in the previous section the number of distinct generated $Q(L)$ is finite, and the computation always stops. Moreover the generated consequences are relevant to the initial query, since the generated sub-queries are directly, or indirectly, related to the initial query.

However the following example shows that its efficiency can be improved. If

MEDB is:

$$\text{MEDB} = \{ Q(p(a,x)), \text{Ax}(p(x,y) \leftarrow q(x,z) \wedge r(z,y)), \text{Ax}(q(a,b_1)), \text{Ax}(r(b_1,c)), \text{Ax}(r(b_2,c)), \dots, \text{Ax}(r(b_n,c)) \}$$

the computation of I_{MDB} generates:

1. $Q(q(a,z)), Q(r(z,y))$
2. $\text{Th}(q(a,b_1)), \text{Th}(r(b_1,c)), \text{Th}(r(b_2,c)), \dots, \text{Th}(r(b_n,c))$
3. $\text{Th}(p(a,c))$

It is clear that a sub-query more specific than $Q(r(z,y))$ can be generated if the answer $\text{Th}(q(a,b_1))$ to the query $Q(q(a,z))$ is reused to generate sub-query $Q(r(b_1,y))$, whose answer is: $\text{Th}(b_1,c)$, and this avoids to generate the set of useless answers: $\text{Th}(r(b_2,c)), \dots, \text{Th}(r(b_n,c))$. This very simple observation has motivated the definition of the next strategy.

Features:

- Valid and Complete (with respect to the derivation of ground atomic consequences of DB).
- Efficiency: much more better than Forward chaining, less efficient than Backward chaining in the cases where it terminates.
- Termination: yes.

3.4 Sequential Backward-Forward chaining

This strategy is described by the following MIDB:

- (0) $Q(l) \wedge \text{Ax}(l) \rightarrow \text{Th}(l)$
- (1) $Q(l) \wedge \text{Ax}(l \leftarrow l_1 \wedge \dots \wedge l_n) \rightarrow Q(l_1)$
-
- (i) $Q(l) \wedge \text{Ax}(l \leftarrow l_1 \wedge \dots \wedge l_{i-1} \wedge l_i \wedge \dots \wedge l_n) \wedge \text{Th}(l_1) \wedge \dots \wedge \text{Th}(l_{i-1}) \rightarrow Q(l_i)$
-
- (n) $Q(l) \wedge \text{Ax}(l \leftarrow l_1 \wedge \dots \wedge l_n) \wedge \text{Th}(l_1) \wedge \dots \wedge \text{Th}(l_n) \rightarrow \text{Th}(l)$

The form of the rule (i) clearly shows that a sub-query on l_i is generated only if answers have been found for sub-queries corresponding to l_1, l_2, \dots, l_{i-1} , and that the sub-query on l_i can take advantage of the instantiation of the variables of L_{DB} that l_i shares with l_1 , or l_2 , or \dots or l_{i-1} .

The computation of I_{MDB} with the MEDB presented in the previous section:

$$\text{MEDB} = \{ Q(p(a,x)), \text{Ax}(p(x,y) \leftarrow q(x,z) \wedge r(z,y)), \text{Ax}(q(a,b_1)), \text{Ax}(r(b_1,c)),$$

... ,Ax(r(b_n,c)) }

generates:

1. Q(q(a,z))
2. Th(q(a, b₁))
3. Q(r(b₁,y))
4. Th(r(b₁,c))
5. Th(p(a,c))

This strategy is very close to the *standard* Prolog strategy. Nevertheless it is worth noting some significant differences.

The first one is that the generation of consequences using the Least Fixpoint operator I_{MDB} allows to check the generation of sub-queries that have still been generated in some previous computation steps. The consequence is that there is no duplicated computations, as it may be the case in Prolog, and the computation always terminates.

The second one is that it is possible to refine the definition of the strategy in such a way that the order of the l_i s in $Ax(l \leftarrow l_1 \wedge \dots \wedge l_n)$ depends on L_{DB} variables instantiated in $Q(L)$. In the rule (i) some of the variables in l are instantiated and all the variables that appear in l_1 or l_2 or ... or l_{i-1} are instantiated. So, the ordering is defined in such a way that l_i shares the maximum number of variables with the instantiated variables in l or l_1 or l_2 or ... or l_{i-1} . For instance, if we consider the example of the ancestors:

$$Ax(anc(x,y) \leftarrow anc(x,z) \wedge parent(z,y))$$

and the query: $Q(anc(x,a))$, the literals must be reordered as follows:

$$Ax(anc(x,y) \leftarrow parent(z,y) \wedge anc(x,z))$$

Then the corresponding instance of the rule (1) is :

$$(1) Q(anc(x,a)) \wedge Ax(anc(x,a) \leftarrow parent(z,a) \wedge anc(x,z)) \rightarrow Q(parent(z,a))$$

an example of instance of the rule (2) is:

$$(2) Q(anc(x,a)) \wedge Ax(anc(x,a) \leftarrow parent(b,a) \wedge anc(x,b)) \wedge Th(parent(b,a)) \rightarrow Q(anc(x,b))$$

and an example of instance of the rule (3) is:

$$(3) Q(anc(c,a)) \wedge Ax(anc(c,a) \leftarrow parent(b,a) \wedge anc(c,b)) \wedge Th(parent(b,a)) \wedge Th(anc(c,b)) \rightarrow Th(anc(c,a))$$

This reordering process is in fact one of the main features of strategies like: ALEXANDRE [23, 24, 14], MAGIC SETS [3, 26, 2], or QSQ [28, 27]. Indeed in

these strategies the order of generated sub-queries depends on the mode of the arguments of the initial query. The mode of an argument is: “bounded”, if it is a constant or a bounded variable, and “unbounded”, in other cases. The subqueries are ordered to have a maximum number of arguments of bounded mode for each sub-query. Reasoning on the mode of the arguments allows to define the ordering independently of the particular values of the constants, and also allows to “compile” the rule depending on the mode of the arguments. The “compilation” technique is quite different for each author, but the intuitive idea is the same.

In a preliminary extended version of this paper we have extended the presented framework in order to define the “compilation” process itself in terms of meta rules. The basic idea is to consider an abstract interpretation where all the distinct constants of L_{DB} are abstracted in a unique constant denoted by 1. Variables names at the abstract level are irrelevant in the sense that two rules, or two queries that are variant one each other are considered as the same rule or as the same query. The set of facts of EDB is abstracted in a unique fact for each predicate symbol. In our example we should have: $\text{parent}(1,1)$ and $\text{anc}(1,1)$. Instances of the rules (1), (2) and (3) take at the abstract level the form:

$$(1) \text{Q}(\text{anc}(x, 1)) \wedge \text{Ax}(\text{anc}(x, 1) \leftarrow \text{parent}(z, 1) \wedge \text{anc}(x, z)) \rightarrow \text{Q}(\text{parent}(z, 1))$$

$$(2) \text{Q}(\text{anc}(x, 1)) \wedge \text{Ax}(\text{anc}(x, 1) \leftarrow \text{parent}(1, 1) \wedge \text{anc}(x, 1)) \wedge \text{Th}(\text{parent}(1, 1)) \rightarrow \text{Q}(\text{anc}(x, 1))$$

$$(3) \text{Q}(\text{anc}(1, 1)) \wedge \text{Ax}(\text{anc}(1, 1) \leftarrow \text{parent}(1, 1) \wedge \text{anc}(1, 1)) \wedge \text{Th}(\text{parent}(1, 1)) \wedge \text{Th}(\text{anc}(1, 1)) \rightarrow \text{Th}(\text{anc}(1, 1))$$

At the abstract level $\text{Q}(\text{parent}(z,1))$ and $\text{Q}(\text{anc}(x,1))$ are query types. For instance the queries $\text{Q}(\text{anc}(x,a))$ and $\text{Q}(\text{anc}(y,b))$ are of the type $\text{Q}(\text{anc}(x,1))$.

Then we define new meta predicates to represent links between the object level and the abstract level. These new predicates are used to define three sets of rules of L_{MDB} . The first set allows to derive all the query types corresponding to all the object query that have to be answered to get the answer to the initial query. For instance to compute the answers to $\text{Q}(\text{anc}(x,a))$ we have to compute answers to queries of types $\text{Q}(\text{parent}(z,1))$ and $\text{Q}(\text{anc}(x,1))$. The second set allows to generate transformed form of the rules in IDB depending on the query types we have to answer. For instance for queries of type $\text{Q}(\text{anc}(x,1))$ the generated rules corresponding to the axiom $\text{Ax}(\text{anc}(x, y) \leftarrow \text{parent}(z, y) \wedge \text{anc}(x, z))$ are:

$$(1') \text{T}(\text{anc}(x, y), \text{anc}(x, 1)) \wedge \text{Q}(\text{anc}(x, y)) \rightarrow \text{Q}(\text{parent}(z, y))$$

$$(2') \text{T}(\text{anc}(x, y), \text{anc}(x, 1)) \wedge \text{Q}(\text{anc}(x, y)) \wedge \text{Th}(\text{parent}(z, y)) \rightarrow \text{Q}(\text{anc}(x, z))$$

$$(3') \text{T}(\text{anc}(x, y), \text{anc}(x, 1)) \wedge \text{Q}(\text{anc}(x, y)) \wedge \text{Th}(\text{parent}(z, y)) \wedge \text{Th}(\text{anc}(x, z)) \rightarrow \text{Th}(\text{anc}(x, y))$$

where $T(\text{anc}(x,y),\text{anc}(x,1))$ can be read “ $\text{anc}(x,y)$ is of type $\text{anc}(x,1)$ ”. So, if we consider an instance of the rule (1') corresponding to $Q(\text{anc}(x,a))$, we have to evaluate $T(\text{anc}(x,a),\text{anc}(x,1))$, which is evaluated to true. Finally a third set allows to compute answers obtained from the transformed rules.

Features:

- Valid and Complete (with respect to the derivation of ground atomic consequences of DB).
- Efficiency: the most efficient among the strategies that always terminate.
- Termination: yes.

4 Non Horn Clauses

Deductive Databases can be extended to Non Horn clauses [16, 21, 22] to cover new applications where the extensional part may contain disjunctive information, or to integrate in the derivation process Integrity Constraints that are not necessarily Horn clauses.

The form of the answers can also be extended to Intensional Answers or to Conditional Answers. If a query is represented by a literal l of L_{DB} these answers take the form: $l' \leftarrow h$, where l' is an instance of l , such that $l' \leftarrow h$ is a consequence of IDB, in the case of Intensional Answers [5], or ground consequences of DB, in the case of Conditional Answers [7]. Additional constraints on these types of answers are that $l' \leftarrow h$ must not be a tautology, and it must be minimal with respect to subsumption. For simplification these two constraints will be ignored in the presentation of the strategies.

4.1 Backward chaining

This strategy is described by the following MIDB:

- (1) $Q(l) \wedge Ax(l \leftarrow h) \rightarrow Th(l \leftarrow h)$
- (2) $Q(l) \wedge Th(l \leftarrow l_1 \wedge h_1) \wedge \dots \wedge Th(l \leftarrow l_i \wedge h_i) \wedge Ax(l_1 \vee \dots \vee l_i \leftarrow l) \rightarrow Th(l \leftarrow h_1 \wedge \dots \wedge h_i \wedge h)$
- (3) $M(l \vee l \vee c) \rightarrow M(l \vee c)$

where M can be either Ax or Th .

The rule (2) generalizes the corresponding rule for Horn clauses in section 3.1. It corresponds to an hyperresolution where the nucleus is in an Input set.

The rule (3) corresponds to factorization. It is necessary for completeness. For example this rule allows to generate: $\text{Ax}(p(x) \vee \neg q(x, x))$ from:
 $\text{Ax}(p(x) \vee p(y) \vee \neg q(x, y))$.

Let's take an example where this strategy is used to compute Intensional Answers. We consider an IDB about properties of partners of an Esprit project:

$\text{industrial}(x) \vee \text{research.lab}(x) \leftarrow \text{partner}(x)$
 $\text{industrial}(x) \leftarrow \text{prime}(x)$
 $\text{main.partner}(x) \leftarrow \text{industrial}(x)$
 $\text{main.partner}(x) \leftarrow \text{research.lab}(x) \wedge \text{large}(x)$

Let's consider MEDB that only contains the axioms of IDB and the query:
 $Q(\text{main.partner}(x))$.

The computation of I_{MIDB} generates with rule (1):

1. $\text{Th}(\text{main.partner}(x) \leftarrow \text{industrial}(x))$,
 $\text{Th}(\text{main.partner}(x) \leftarrow \text{research.lab}(x) \wedge \text{large}(x))$

In a second step, using rule(2), we get:

2. $\text{Th}(\text{main.partner}(x) \leftarrow \text{prime}(x))$,
 $\text{Th}(\text{main.partner}(x) \leftarrow \text{partner}(x) \wedge \neg \text{research.lab}(x))$

Using both consequences generated in 1. , from the rule (2) we get:

3. $\text{Th}(\text{main.partner}(x) \leftarrow \text{partner}(x) \wedge \text{large}(x))$

There is no guarantee of termination for the same reason as for Horn clauses.

Features:

- Valid and Complete (with respect to the derivation of clauses that contain an instance, in L_{DB} , of the literal of the query [8]).
- Efficiency: excellent when the computation syops.
- Termination: not necessarily.

4.2 Sequential Backward Forward chaining

Since Conditional Answers are defined as ground clauses of L_{DB} , any Conditional Answer contains a finite number of clauses, and it should be possible to define a strategy to compute these answers which terminates. We present here a strategy that has this property, but it is an open question to know if it is complete or not. We have to define the new meta predicates:

$\text{GAx}(x)$: x is a ground instance, in L_{DB} , of an axiom in DB.

$EAx(x)$: x is an axiom of EDB.

The strategy is described by the following MIDB:

$$(0) \quad Q(l) \wedge EAx(l \leftarrow h) \rightarrow Th(l \leftarrow h)$$

$$(1) \quad Q(l) \wedge Ax(l \leftarrow l_1 \wedge \dots \wedge l_n) \rightarrow Q(l_1)$$

....

$$(i) \quad Q(l) \wedge Ax(l \leftarrow l_1 \wedge \dots \wedge l_{i-1} \wedge l_i \wedge \dots \wedge l_n) \wedge Th(l_1 \leftarrow h_1) \wedge \dots \wedge Th(l_{i-1} \leftarrow h_{i-1}) \rightarrow Q(l_i)$$

....

$$(n) \quad Q(l) \wedge GAx(l \leftarrow l_1 \wedge \dots \wedge l_n \wedge h) \wedge Th(l_1 \leftarrow h_1) \wedge \dots \wedge Th(l_n \leftarrow h_n) \rightarrow Th(l \leftarrow h_1 \wedge \dots \wedge h_n \wedge h)$$

$$(f) \quad M(l \vee l \vee c) \rightarrow M(l \vee c)$$

It is similar to the strategy defined for Horn clauses in section 3.4. The main difference is that facts in EDB are, in some sense, conditional facts, that is, they are of the form: $l_i \leftarrow h_i$ instead of l_i . So, they are represented in L_{MDB} by atoms of the form $EAx(l_i \leftarrow h_i)$ instead of $Ax(l_i)$, in the rule (0). For the same reason, generated consequences are of the form $Th(l_i \leftarrow h_i)$ instead of $Th(l_i)$.

Since axioms in EDB are not necessarily atomic, to distinguish these axioms and those in IDB we need to use in the rule (0) the meta predicate EAx instead of Ax .

Ground atoms that satisfy $EAx(l \leftarrow h)$ in the rule (0) correspond to ground clauses of EDB, then the generated $Th(l \leftarrow h)$ also correspond to a ground clause of L_{DB} .

In the rule (n) if all the atoms of the form $Th(l_i \leftarrow h_i)$ correspond to ground clauses of L_{DB} the generated consequence has the same property only in the case where we impose the constraint that: $l \leftarrow l_1 \wedge \dots \wedge l_n \wedge h$ is instantiated by a ground clause of L_{DB} . That is the constraint imposed by the predicate GAx . Then we can easily prove, by induction, that all the generated consequences correspond to ground clauses of L_{DB} , and that the computation always terminates.

Let's consider an example where we have in MEDB: $Ax(p(x) \leftarrow q(x) \wedge r(x, y))$, $EAx(q(a) \leftarrow s(a))$, $EAx(r(a, b) \leftarrow t(a, b))$ and $Q(p(x))$. Even if $Th(q(a) \leftarrow s(a))$ is generated, the ground instance of the rule (n):

$$Q(p(a)) \wedge GAx(p(a) \leftarrow q(a) \wedge r(a, y)) \wedge Th(q(a) \leftarrow s(a)) \rightarrow Th(p(a) \rightarrow s(a) \wedge r(a, y))$$

is not satisfied because $p(a) \leftarrow q(a) \wedge r(a, y)$ is not ground. However if $Th(r(a, b) \leftarrow t(a, b))$ is generated the ground instance of the rule (n):

$$Q(p(a)) \wedge GAx(p(a) \leftarrow q(a) \wedge r(a, b)) \wedge Th(r(a, b) \leftarrow t(a, b)) \rightarrow Th(p(a) \rightarrow q(a) \wedge t(a, b))$$

is satisfied, since $p(a) \leftarrow q(a) \wedge r(a, b)$ is ground.

At the present time we have no denotational definition of what is a Conditional answer. The problem comes from the fact that we want to characterize answers that are ground clauses of L_{DB} , and that are minimal for subsumssion. But, for some examples, it seems to be too strong to impose both constraints. For instance if we have in IDB: $p(x) \leftarrow q(x)$ and the query: $p(a)$, we would like to get the Conditional answer: $p(a) \leftarrow q(a)$, though it is not minimal for subsumssion.

Features:

- Valid, Complete(open question).
- Efficiency: good.
- Termination: yes.

5 Conclusion

We have presented a general framework that has some typical features of Relational Databases since the answers are not computed “tuple at a time” like in Prolog, but by sets. This is reflected in the definition of the Least Fixpoint operator I_{MDB} .

The representation of the rules in IDB does not express information about the computation control. However some kind of high level control can be defined at the meta level. We have seen how the paradigm of abstract interpretation can be used to select the literals to be resolved in a clause.

Moreover termination is a strong concern. The termination can be guaranteed for several strategies, because all the sub-queries are memorized, and because we have defined an equivalence relation on queries (variants).

The last common feature is that, in the context of non Horn clauses, negation has its pure logical meaning.

We have also seen that the Sequential Backward Forward chaining strategy is very close to the Prolog strategy (SLD resolution). A significant difference is that we compute sets of theorems, instead of sets of substitutions. This allows an easy extension to answers that are not sets of atoms but sets of clauses, as we have seen in section 4. An interesting further work would be to reformulate existing extensions of SLD resolution to non Horn clauses, like SLI resolution [16] or SOL resolution [12, 13], and to analyze their similarities and differences.

Appendix

Property2: Continuity of T_{MDB}

The operator T_{MDB} is continuous.

Proof:

Let S be a subset of $2^{\text{H}_{\text{MDB}}}$ we have to prove:

$$T_{\text{MDB}}(\text{lub}(S)) = \text{lub}(\{T_{\text{MDB}}(s) : s \in S\})$$

where $\text{lub}(S)$ denotes the least upper bound of S .

Let's call Σ the set: $\{T_{\text{MDB}}(s) : s \in S\}$.

We first prove that $T_{\text{MDB}}(\text{lub}(S))$ is included in $\text{lub}(\Sigma)$.

From the definition of the operator T_{MDB} , $\text{mf} \in T_{\text{MDB}}(\text{lub}(S))$ implies that we have (1) $\text{mf} \in \text{lub}(S)$, or (2) $\text{mf} \in \text{MEDB}$, or there exists an object instance $\{\text{mf}_1, \text{mf}_2, \dots, \text{mf}_n\}$ of $\{\text{mf}'_1, \text{mf}'_2, \dots, \text{mf}'_n\}$, and a ground instance of a rule in MIDB of the form: $\text{mf}_1 \wedge \text{mf}_2 \wedge \dots \wedge \text{mf}_n \rightarrow \text{mf}$, such that this object instance satisfies the rule, and $\{\text{mf}'_1, \text{mf}'_2, \dots, \text{mf}'_n\}$ is included in $\text{lub}(S)$.

The proposition (1) or (2) implies that there exists some s in S such that mf is in s or mf is in MEDB . Therefore, by definition of the Closure operator mf is in $T_{\text{MDB}}(s)$, and then mf is in $\text{lub}(\Sigma)$.

From (3), since we have $\{\text{mf}'_1, \text{mf}'_2, \dots, \text{mf}'_n\}$ included in $\text{lub}(S)$, and this set is finite, there exists a set s' in S such that $\{\text{mf}'_1, \text{mf}'_2, \dots, \text{mf}'_n\}$ is included in s' . Then mf is in $T_{\text{MDB}}(s')$, and therefore mf is in $\text{lub}(\Sigma)$.

Now we prove that $\text{lub}(\Sigma)$ is included in $T_{\text{MDB}}(\text{lub}(S))$.

If mf is in $\text{lub}(\Sigma)$ there exists a set s in S such that mf is in $T_{\text{MDB}}(s)$. Then we have (1) mf is in s , or (2) mf is in MEDB , or (3) there exists an object instance $\{\text{mf}_1, \text{mf}_2, \dots, \text{mf}_n\}$ of $\{\text{mf}'_1, \text{mf}'_2, \dots, \text{mf}'_n\}$, and a ground instance of a rule in MIDB of the form: $\text{mf}_1 \wedge \text{mf}_2 \wedge \dots \wedge \text{mf}_n \rightarrow \text{mf}$, such that this object instance satisfies the rule, and $\{\text{mf}'_1, \text{mf}'_2, \dots, \text{mf}'_n\}$ is included in s .

If mf is in s , mf is also in $\text{lub}(S)$, then proposition (1) or (2) implies that mf is in $T_{\text{MDB}}(\text{lub}(S))$.

Since $\{\text{mf}'_1, \text{mf}'_2, \dots, \text{mf}'_n\}$ is also included in s , $\{\text{mf}'_1, \text{mf}'_2, \dots, \text{mf}'_n\}$ is included in $\text{lub}(S)$, and therefore mf is in $T_{\text{MDB}}(\text{lub}(S))$.

Property4: Correspondance between the Operators I_{MDB} and T_{MDB}

Let S_n be defined by: $S_n = T_{\text{MDB}}^n(\emptyset)$, then we have: $(S_n, S_n - S_{n-1}) = I_{\text{MDB}}^n(\emptyset, \emptyset)$.

Proof: The proof is by induction on n .

For $n=1$, we have $T_{\text{MDB}}(\emptyset) = \text{MEDB}$, then the property is a direct consequence of the definition of the operator I_{MDB} .

Assume the property holds for each i less or equal than n . Then we have $S'_n = S_n$, $D_n = S_n - S_{n-1}$. If we have: $(S'_{n+1}, D_{n+1}) = I_{\text{MDB}}(S_n, D_n)$, we have to prove that: $S'_{n+1} = S_{n+1}$, and $D_{n+1} = S_{n+1} - S_n$.

By definition of T_{MDB} , $\text{mf} \in S_{n+1}$ is equivalent to $\text{mf} \in T_{\text{MDB}}(S_n)$, which is equivalent to (1) $\text{mf} \in S_n$ or (2) $\text{mf} \in S_1$ or (3) there exists a ground instance of a rule r in MIDB :

$\text{mf}_1 \wedge \text{mf}_2 \wedge \dots \wedge \text{mf}_n \rightarrow \text{mf}$

such that: $\{\text{mf}_1, \text{mf}_2, \dots, \text{mf}_n\}$ is an object instance of $\{\text{mf}'_1, \text{mf}'_2, \dots, \text{mf}'_n\}$ that satisfies r , and $\{\text{mf}'_1, \text{mf}'_2, \dots, \text{mf}'_n\}$ is included in S_n , and there exists some mf'_i that is in D_n .

By definition of D_n , $\text{mf}'_i \in S_n$ is equivalent to $\text{mf}'_i \in S_{n-1}$ or $\text{mf}'_i \in D_n$.

Therefore, either all the mf'_i 's are in S_{n-1} , and, by definition of T_{MDB} , mf is in S_n , or there exists some j in $[1, n]$ such that mf'_j is in D_n , and, by definition of I_{MDB} , mf is in N_{n+1} . Then we have $\text{mf} \in S_n$ or $\text{mf} \in N_{n+1}$. Since we have: $S'_{n+1} = S_n \cup N_{n+1}$, mf is in S'_{n+1} . Then $\text{mf} \in S_{n+1}$ is equivalent to $\text{mf} \in S'_{n+1}$, and we have $S_{n+1} = S'_{n+1}$.

By definition of I_{MDB} we have: $D_{n+1} = S'_{n+1} - S'_n$. Since we have: $S'_{n+1} = S_{n+1}$ and $S'_n = S_n$, we have: $D_{n+1} = S_{n+1} - S_n$.

References

- [1] S. Abramski and C. Hankin. An Introduction to Abstract Interpretation. In S. Abramski and C. Hankin, editors, *Abstract Inerpretation of Declarative Languages*. Ellis Horwood Ltd, 1987.
- [2] F. Bancilhon, D. Mayer, Y. Sagiv, and J. Ullman. Magic sets and other strange ways to implement logic programmms. In *Proc. 5th ACM Symposium on Principles of Database Systems*, 1986.
- [3] C. Beeri and R. Ramakrishnan. On the Power of Magic. *The Journal of Logic Programming*, 10(3-4), 1991.
- [4] F. Bry. Query evaluation in Deductive Databases: bottom-up and top-down reconciled. *Data and Knowledge Engineering*, 25(4), 1990.
- [5] L. Cholvy and R. Demolombe. Querying a Rule Base. In *Proc. of 1st Int. Conf. on Expert Database Systems*, 1986.

- [6] R.C.T. Lee C.L. Chang. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.
- [7] R. Demolombe. A Strategy for the Computation of Conditional Answers. In *10th European Conference on Artificial Intelligence*, 1992.
- [8] R. Demolombe and L. Fariñas del Cerro. An Inference Rule for Hypothesis Generation. In *Proc. of International Joint Conference on Artificial Intelligence*, Sydney, 1991.
- [9] R. Demolombe and V.Royer. Evaluation strategies for recursive axioms : a uniform presentation. In *Journées FIRTECH : Bases de Données et Intelligence Artificielle*, 1987.
- [10] G. Grahne, S. Sippu, and E. Soisalon-Soininen. Efficient Evaluation for a Subset of Recursive Queries. *The Journal of Logic Programming*, 10(3-4), 1991.
- [11] L.J. Henschen and S.A. Naqvi. On Compiling Queries in Recursive First-Order Databases. *Journal of ACM*, 31,1, 1984.
- [12] K. Inoue. Linear Resolution for Consequence Finding. *Artificial intelligence, an International Journal*, 56, 1992.
- [13] K. Inoue. *Studies on Abductive and Nonmonotonic Reasoning*. PhD thesis, Kyoto University, 1992.
- [14] J-M. Kerisit. *La methode Alexandre : une technique de deduction*. PhD thesis, Universite de Paris7, 1988.
- [15] M. Kifer and E. Lozinskii. Filtering Data Flow in Deductive Databases. In *Proc. International Conference on Database Theory*, 1986.
- [16] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, 1992.
- [17] E. Lozinski. Evaluating queries in deductive databases by generating. In *Proc of 11th International Joint Conference on Artificial Intelligence*, 1985.
- [18] A. Marchetti-Spaccamela, A. Pelaggi, and D. Sacca. Comparison of Methods for Logic-Query Implementation. *The Journal of Logic Programming*, 10(3-4), 1991.
- [19] D. McKay and S. Shapiro. Using Active Connection Graphs for Reasoning with Recursive Rules. In *Proc. 7th Int. Joint Conf. on Artificial Intelligence*, Vancouver, 1981.

- [20] C. Mellish. Abastract Interpretation of Prolog Programs. In S. Abramski and C. Hankin, editors, *Abstract Inerpretation of Declarative Languages*. Ellis Horwood Ltd, 1987.
- [21] J. Minker and A. Rajasekar. A fixpoint semantics for non-horn logic programs. Technical Report UMIACS-TR-87-24, University of Maryland, IACS, 1987.
- [22] J. Minker and A. Rajasekar. Procedural interpretation of non-horn logic programs. In *Proc. Conference on Automated Deduction*, 1988.
- [23] J. Rohmer and R. Lescoeur. La Méthode Alexandre: Une solution pour traiter les Axiomes Récursifs dans les Bases de Donnés Dductives. In *Colloque Reconnaissance des Formes et Intelligence Artificielle*, 1985.
- [24] J. Rohmer, R. Lescoeur, and J-M. Kerisit. The alexander method : a technique for the processing of recursive axioms in deductive databases. *New Generation Computing*, Vol. 4(Num. 3), 1986.
- [25] D. Sacca and C. Zaniolo. The Generalized Counting Method for Recursive Logic Queries. In *Proc. International Conference on Database Theory*, 1986.
- [26] J. Ullman. Implementation of Logical Query Languages for Databases. *ACM Transactions On Database Systems*, 10(3), 1985.
- [27] L. Vieille. From QSQ to QoSAQ: Global Optimization of Recursive Queries . In *Proc. 2nd Int. Conf. on Expert Database Systems*, 1987.
- [28] L. Vieille. Recursive axioms in deductive databases : the query-sub-query approach. In L.Kerschberg, editor, *Proc. 1st Int. Conf. on Expert Database Systems*. Benjamin/Cummings Pub. Comp., 1987.