Strategies for the computation of Conditional Answers *

Robert Demolombe
ONERA/CERT
2 avenue E.Belin B.P. 4025
31055 Toulouse
France

September 1990

Abstract

We consider here non-Horn Deductive Data Bases (DDB). In this context there are many queries whose answer is : *I don't know.* A first approach to reduce the number of such answers is to add information, like default rules, in order to automatically generate assumptions. The second approach, which is adopted in this paper, is to provide to the user the conditions that guarantee the validity of the answer. These conditional answers are generated by standard reasoning, and not by default reasoning.

Then the problem is the following : if T represents the DDB and q the query, and if there is no direct answer to q, we want to produce the more general conditions c such that : $T \vdash q \leftarrow c$. We present two strategies, GASP and GALP, designed for this purpose. They are defined by meta rules, and the meta rules can be used for a least fixpoint operator definition. We show that the GASP strategy is always more efficient than GALP, but the GALP strategy can be adapted in order to compute ground conditional answers. The least fixpoint operator associated to GRALP (the strategy adapted from GALP) computes the answer in a finite number of steps, even if the DDB contains recursive definitions.

*This work has been partially suported by the CEC, in the context of the Basic Research Action, called MEDLAR.

1 Introduction

Many works have been devoted to the *standard* approach of Deductive Data Bases (DDB) [1, 12, 5, 13]. In this approach a DDB is composed of two parts : a set of rules, the Intensional Data Base (IDB), which is a set of definite Horn clauses, and a set of facts, the Extensional Data Base (EDB), which is a set of ground atoms.

More recently this approach has been extended to disjunctive DDB where the rules are not necessarily Horn clauses [8, 3, 6] , and facts may be ground positive clauses.

In this paper we extend disjunctive DDB to the case where EDB may contain any kind of ground clauses. But the most significant contribution is to consider answers which are of a different type, and are called *Conditional Answers.* Conditional answers are another way to deal with incompleteness. Indeed the usual appraoch is to complete the DDB with some kind of meta rule like Closed World Assumption (CWA), or Generalised Closed World Assumption (GCWA) [7], in the context of disjunctive DDB, or default rules in the context of non-monotonic reasoning [10]. In the Conditional Answer approach no assumption is added to the DDB by applying some kind of default reasoning. When there is not enough information in the DDB to answer a given query, the answer provide the less restrictive assumptions which allow to infer the query.

Let's consider for example the very simple DDB :

$A \vee B \leftarrow C$ $\qquad\qquad$ C

and the query : A?

In that case we cannot provide a direct answer to the query, but we can provide the conditional answer : $A \leftarrow \neg B$. Then the user knows that A is true under the assumption $\neg B$, and he can take himself the decision to assume $\neg B$ or not.

It is interesting to consider two kinds of conditional answers. They are illustrated by the next example.

IDB :

$\text{At} - \text{home}(x) \leftarrow \text{Sleeping}(x)$

$\text{Teaching}(x) \leftarrow \text{At} - \text{University}(x)$

$\text{At} - \text{university}(x) \vee \text{At} - \text{home}(x) \leftarrow \text{Working}(x)$

$\leftarrow \text{Teaching}(x) \wedge \text{Sleeping}(x)$

EDB :

$\text{Working}(a), \quad \neg\text{Teaching}(b), \quad \text{Sleeping}(c)$

If we consider the query : At-home(x)? we can interpret the query into two different ways : *what are the conditions which guarantee that somebody is at home, independenlty of a particular situation?*, or : *what are the conditions which guarantee, for some particular individuals, that they are at home ?*. In the first case the answer is called an *Intensional Conditional Answer* [2], in the second case it is called an *Extensional Conditional Answer.* From a technical view point the intensional conditional answers are infered only from IDB, while the extensional conditional answers are infered from IDB and EDB. For that example we have :

Intensional Conditional Answer :

$\text{At} - \text{home}(x) \leftarrow \text{Sleeping}(x)$

$\text{At} - \text{home}(x) \leftarrow \text{Working}(x) \wedge \neg\text{At} - \text{university}(x)$

$\text{At} - \text{home}(x) \leftarrow \text{Working}(x) \wedge \neg\text{Teaching}(x)$

These answers are fomulas F(x) such that :

$\text{IDB} \vdash \text{At} - \text{home}(x) \leftarrow F(x)$

Extensional Conditional Answer :

$\text{At} - \text{home}(a) \leftarrow \neg\text{At} - \text{university}(a)$

$\text{At} - \text{home}(a) \leftarrow \neg\text{Teaching}(a)$

$\text{At}-\text{home}(b) \leftarrow \text{Working}(b)$

These answers are ground formulas F(a) such that :

$\text{IDB} \cup \text{EDB} \vdash \text{At}-\text{home}(a) \leftarrow F(a)$

In the next section is presented a general formal definition of conditional answers holding for the two kinds of answers. Then we present two strategies to compute conditional answers. We compare their relative efficiency, and we point out the particular problem of infinite answers. In the last section we propose a modification to one strategy to compute extensional conditional answers in a finite number of steps.

2 General definition of Conditional Answers

We consider queries which are positive literals. This assumption does not restrict generality. Indeed, if the query is a general formula F(x), we define a new predicate symbol q(x), we add to the DDB the formula $Q = (q(x) \leftarrow F(x))\forall x$, and the query is represented by the positive literal q(x).

EDB is a set of ground formulas. IDB is a set of formulas. We consider the theory $T = \text{IDB}\cup\text{EDB}\cup Q$, where all the formulas are represented in clausal form, and each clause is Range Restricted; that is, if a variable occurs in a positive literal in a clause, it must also occur in a negative literal of that clause. The clauses are considered as sets of literals. Moreover we consider clauses *without functional symbols.*

Conditional Answer Definition

Let q be a positive literal, the conditional answer to the query q is the set of clauses :

$\{\ q\sigma \vee c \mid T \vdash q\sigma \vee c$ and $q\sigma \vee c$ is not a tautology, and $q\sigma \vee c$ is minimal wrt subsumption $\}$

A clause c is minimal with regard to subsumption, in the context of T, if there is no clause c' derivable from T such that c' subsumes c.

A clause c' subsumes a clause c if there exists a substitution σ such that : $c'\sigma \subseteq c$.

The clause c is called by Reiter and de Kleer, in [11], a minimal support for $q\sigma$. The clauses $q\sigma \vee c$ satisfying these properties are called minimal implicants.

The intuitive motivations for such definition are discussed below.

Notice first that a clause $q\sigma \vee c$ can be alternatively be represented by the formula $q\sigma \leftarrow \neg c$. If $q\sigma \vee c$ is a tautology c contains $\neg q\sigma$, and the clause can be represented by the formula $q\sigma \leftarrow q\sigma \wedge c'$. Since the condition is stronger than the query itself it is out of interest.

If $q\sigma \vee c$ is minimal wrt subsumption c is not a theorem, and the assumption $\neg c$ is consistent with T. Moreover $\neg c$ is the less restrictive assumption. Indeed there is no clause derivable from T, of the form : $q\sigma \leftarrow \neg c'$, such that $\neg c \rightarrow \neg c'$. Another consequence of the minimality is that $q\sigma$ is not a theorem, that is, we need some assumption to guarantee the validity of $q\sigma$.

Extended Conditional Answer Definition

With the same notations we define an extended conditional answer as the set of clauses :

{ $q\sigma \vee c$ | $T \vdash q\sigma \vee c$ and $q\sigma \vee c$ and there is no clause c' such that : $T \vdash q\sigma \vee c'$ and c' subsumes c }

We can easily see that, for a given query, the extended conditional answer contains the conditional answer. The only difference is that for clauses in the extended conditional answer there is no guarantee that c is not a theorem of T; that means $\neg c$ may be an inconsistent assumption.

The definition of extended conditional answers is introduced only for technical matters because the strategies presented in the next sections compute just extended conditional answers. If the user want to know if some assumption is inconsistent we can apply, in a further computation step, standard theorem proving strategies. We does not consider that case in this work in order to concentrate on problems which are specific to conditional answers.

3 Intuition of the two strategies

The two strategies presented in this section have been specifically designed to compute extended conditional answers. They work as well for intensional answer as for extensional answers. They are both based on two inference rules, presented in [4], which can be defined in reference to Resolution Principle.

They are informally described here using the example of the first section. For this description we call *relevant theorem* for a given query, a clause derivable from T containing the query, or one of its instances.

The first strategy is called GASP, an abreviation for Generate As Soon as Possible. In this strategy we try to generate relevant theorems for the initial query by resolving one, or several, relevant theorems, previously generated, with a given axiom.

The second strategy is called GALP, an abreviation for Generate As Late as Possible. In this strategy axioms which are relevant for a given query are used to generate subqueries; later the answers to these subqueries are resolved with this axiom to generate relevant theorem for the query. The process starts with the initial query, and recursively generates subqueries and their relevant theorems.

The theory T after transformation in clausal form give :

(1) $Hx \vee \neg Sx$ (2) $Tx \vee \neg Ux$ (3) $Ux \vee Hx \vee \neg Wx$ (4) $\neg Tx \vee \neg Sx$
(5) Wa (6) $\neg Tb$ (7) Sc

where : H, U, W, T and S are repectively abreviations for : At-home, At-university, Working, Teaching and Sleeping; for simplicity P(x) is written Px.

Queries are represented with the question mark as usual.

GASP computation :

Step1 : (8) Hx?

Step2 : (9) $Hx \vee \neg Sx$ (10) $Ux \vee Hx \vee \neg Wx$

Step3 : (11) Hc (12) $Tx \vee Hx \vee \neg Wx$ (13) $Ua \vee Ha$

Step4 : (14) $\neg Sx \vee Hx \vee \neg Wx$ (15) $Ta \vee Ha$ (16) $Hb \vee \neg Wb$

The first relevant theorems (9) and (10), generated in Step2, are the axioms containing the query, or one of its instances. New relevant theorems are generated by resolving (9) or (10) with an axiom. Notice that the resolved literal is always different than the query. This property focus the derivation on relevant theorems. The clause (14) should be removed because it is subsumed by (1).

GALP computation :

Step1 : (8) Hx?

Step2 : (9) $Hx \vee \neg Sx$ (10) Sx? (11) $Ux \vee Hx \vee \neg Wx$ (12) $\neg Ux$? (13) Wx?

Step3 : (14) Sc (15) $Tx \vee \neg Ux$ (16) $\neg Tx$? (17) Wa

Step4 : (18) Hc (19) $Tx \vee Hx \vee \neg Wx$ (20) $Ta \vee Ha$ (21) $Ua \vee Ha$
(22) $\neg Tx \vee \neg Sx$ (23) $\neg Tb$ (24) Sx?

Step5 : (25) $\neg Ux \vee \neg Sx$ (26) $\neg Ub$ (27) Sc

Step6 : (28) $\neg Sa \vee Ha$ (29) $\neg Sx \vee Hx \vee \neg Wx$ (30) $\neg Tc$ (31) $Hb \vee Wb$

Step7 : (32) $\neg Uc$

Step8 : (33) $Hc \vee \neg Wc$

The first generated relevant theorems are the same as in GASP. The difference is that at the Step2 the subquery (10) is generated from the relevant axiom (1). In the same way the subqueries (12) and (13) are generated from te relevant axiom (3). The reason is that any answer to these subqueries is a clause which can be resolved with (9) or (11), to generate a relevant theorem for the query (8). In the same way, at the Step3, the query (12) generates, with the axiom (2), the subquery (16). Each axiom containing the initial query, or a subquery, is a relevant theorem for this query or subquery, and the relevant theorems for the generated subqueries are resolved with the axiom from which the subqueries were generated. For example (22), which is a relevant theorem for the subquery (16), is resolved with the axiom (2) to generate the relevant theorem (25).

Notice that (20) is generated by an hyperresolution from (3), (15) and (17).

The clauses (25), (28) and (33) should be removed because they are subsumed by previously generated clauses.

A common feature to both strategies is to focus as far as possible on relevant theorems. These theorems are relevant to the initial query in GASP, and are relevant to the queries and subqueries in GALP. Both strategies are based on Resolution Principle, and at each step at least one parent clause is an axiom.

They also allow hyperresolution.

4 Formal definition of the two strategies

In this section the strategies are formally defined by meta rules. These rules express, at a meta level, the derivation control, and they are evaluated by a straighforward method which is an incremental saturation by level. Here incremental means that, when a new sentence is generated by a meta rule, at least one of the premisses in the rule is a new sentence. It is important not to confuse the strategy used for meta rule evaluation, and the derivation strategy, at the object level, which is decribed by these meta rules.

Notations :

Meta-variables :

- q, l_i : literal variable; these variables are instantiated by literals at the object level.
- $\neg l_i$: literal variables; these variables are instantiated by the complementary literal which instantiates l_i.
- c_i : clause variables; these variables are instantiated by set of literals at the object level; this set may be empty.
- $l \vee l_i \vee c_i$: denotes the set of literals : $\{l\} \cup \{l_i\} \cup c_i$.
- $l \vee c_1 \vee \ldots \vee c_n \vee c_0$: denotes the set of literals : $\{l\} \cup c_1 \cup \ldots \cup c_n \cup c_0$.

Meta-predicates :

- Query(l) : we have to find all the clauses derivable from T containing l, or an instance of l.
- Ax(c) : c is an axiom of T.
- Th(c) : c is a theorem of T.

GASP Definition

(1) $\text{Query}(q) \land \text{Ax}(q \lor c) \rightarrow \text{Th}(q \lor c)$

(2) $\text{Query}(q) \land \text{Th}(q \lor l_1 \lor c_1) \land \ldots \land \text{Th}(q \lor l_n \lor c_n) \land \text{Ax}(\neg l_1 \lor \ldots \neg l_n \lor c_0)$
$\rightarrow \text{Th}(q \lor c_1 \lor \ldots \lor c_n \lor c_0)$

One could notice that using dots is not allowed in a formal definition. We have used dots here because it would not be difficult to replace these rules by more heavy recursive definitions without dots.

We define a meta theory MT containing the rules (1) and (2), the sentence Ax(c), for each clause c in T, and the sentence Query(q), where q is the literal denoting the initial query.

The sets of sentences generated by saturation by level are denoted by : S_0, S_1, ... S_i ....

S_0 contains all the sentences derivable in one step by the rules (1) and (2) from MT. A sentence is derivable by a rule if there exist a rule instance whose consequence is this sentence, and all the premisses are satisfied by MT. All the tautologies, and all the sentences subsumed by a sentence in MT or S_0 are removed from S_0. We call ΔS_0 the resulting set of derived sentences.

We define S_{i+1} and ΔS_{i+1} in function of S_i and ΔS_i in the following way. We consider all the sentences derivable by the rules (1) and (2) from MT and S_i and we remove from this set all the tautologies and the sentences subsumed by a sentence in MT or S_i. The resulting set of sentences is called ΔS_{i+1} . Then S_{i+1} is defined by : $S_{i+1} = S_i \cup \Delta S_{i+1}$.

If M is any meta predicate, we say the sentence M(c') subsumes the sentence M(c) iff the clause c' subsumes the clause c.

We say that the premisses of the rule (2) (a similar definition applies to rule (1)) are satisfied by a set of sentence S iff :

- the following set of sentence is in S : Query(Q), $\text{Th}(Q_1 \lor L_1 \lor C_1)$, ..., $\text{Th}(Q_n \lor L_n \lor C_n)$, $\text{Ax}(\neg L'_1 \lor \ldots \lor \neg L'_n \lor C_0)$; where Q, Q_1, L_1, C_1, ... , Q_n, L_n, C_n, C_0 are literals or clauses at the object level,
- there exists a more general unifier σ which is solution of the equations : $Q = Q_1 = \ldots = Q_n$

$L_1 = L'_1 \quad L_2 = L'_2 \quad \ldots \quad L_n = L'_n.$

In that case the instantiation of the meta variables is :

$q = Q\sigma \quad l_i = L_i\sigma \quad \neg l_i = \neg L'_i\sigma \quad c_i = C_i\sigma$

and the generated sentence is $Th(q \vee c_1 \vee \ldots \vee c_n \vee c_0)$.

The equations L=L', where L and L' denote $P(t_1, \ldots, t_p)$ and $P(t'_1, \ldots, t'_p)$, or $\neg P(t_1, \ldots, t_p)$ and $\neg P(t'_1, \ldots, t'_p)$, are short hands for the set of equations :

$t_1 = t'_1 \quad t_2 = t'_2 \quad \ldots \quad t_p = t'_p$

The evaluation of the meta rules will be illustrated by the next example. Let's assume S contains the set of sentences :

$Query(Pxy) \quad Th(Paz \vee Qzt \vee \neg Rzt) \quad Th(Puv \vee \neg Suv) \quad Ax(Saw \vee \neg Qvb \vee \neg Tw)$

We have the equations :

$Pxy = Paz = Puw \quad Qzt = Qvb \quad Suv = Saw$

corresponding to the set of equations :

$x = a \quad y = z \quad a = u \quad z = w \quad z = v \quad t = b \quad u = a \quad v = w$

and the most general unifier is the substitution :

$\sigma = \{a/x, y/z, b/t, a/u, y/v, y/w\}$

Then the meta variable instantiations are :

$q = Pxy\sigma = Paz\sigma = Puv\sigma = Pay$

$l_1 = Qzt\sigma = Qyb \quad \neg l_1 = \neg Qvb\sigma = \neg Qyb$

$l_2 = \neg Suv\sigma = \neg Say \quad \neg l_2 = Saw\sigma = Say$

$c_1 = \{\neg Rzt\sigma\} = \{\neg Ryb\} \quad c_2 = \emptyset \quad c_0 = \{\neg Tw\sigma\} = \{\neg Ty\}$

and the generated sentence is :

$\mathrm{Th}(q \vee c_1 \vee c_2 \vee c_0) = \mathrm{Th}(Pay \vee \neg Ryb \vee \neg Ty)$

Let's consider another example, in Propositional Calculus, where S contains the sentences :

(1) Query(P) (2) $\mathrm{Th}(P \vee Q \vee \neg R)$ (3) $\mathrm{Th}(P \vee \neg S)$ (4) $\mathrm{Ax}(S \vee \neg Q \vee \neg T)$

We can generate the sentences :

(5) $\mathrm{Th}(P \vee \neg R \vee S \vee \neg T)$, from (1), (2) and (4),

(6) $\mathrm{Th}(P \vee \neg Q \vee \neg T)$, from (1), (3) and (4),

(7) $\mathrm{Th}(P \vee \neg R \vee \neg T)$, from (1), (2), (3) and (4).

In that example we can see that if several theorems (here (2) and (3)) may be resolved with the same axiom (here (4)), we have to generate all the sentences derivable from every subset of these theorems. Some of the resulting sentences may be subsumed by other ones (here (7) subsumes (5)), but that is not necessarily the case.

GALP Definition

(1) $\mathrm{Query}(q) \wedge \mathrm{Ax}(q \vee c) \rightarrow \mathrm{Th}(q \vee c)$

For each i in [1,p] :

(2.i) $\mathrm{Query}(l) \wedge \mathrm{Ax}(l \vee \neg l_1 \vee \ldots \vee \neg l_i \vee \ldots \vee \neg l_p) \rightarrow \mathrm{Query}(l_i)$

Endfor;

(2) $\mathrm{Query}(l) \wedge \mathrm{Th}(l_1 \vee c_1) \wedge \ldots \wedge \mathrm{Th}(l_n \vee c_n) \wedge \mathrm{Ax}(l \vee \neg l_1 \vee \ldots \neg l_n \vee c_0)$
$\rightarrow \mathrm{Th}(l \vee c_1 \vee \ldots \vee c_n \vee c_0)$

The rules (2.i) show how subqueries are generated from a given query and an axiom containing some instance of that query. The rule (3) show how relevant theorems for the subqueries allow to generate a relevant theorem for the initial query.

Notice that in the GASP strategy each generated theorem is relevant for some subquery, but they are not necessarily relevant for the initial query.

It is easy to see that the interpretation we have defined for the meta rules defining GASP and GALP provides a definition for a least fixed point operator.

5 Efficiency comparison

We can have a rough idea of the relative efficiency of one strategy with respect to the other by comparing the number of generated sentences.

In this section we compare the two strategy in a particular case which give an idea of how the result can be extended to the general case.

For this paticular case we consider a theory T in Propositional Calculus where all the clauses are binary clauses with one positive literal and one negative literal, like : $A \vee \neg B$. In this particular case sentences of the form : $Ax(A \vee \neg B)$ or $Th(A \vee \neg B)$ are represented by : Ax(A,B) or Th(A,B), where the first argument in Ax(x,y) and Th(x,y) represents the positive literal, and the second argument represents the negative literal.

The meta rules defining GASP and GALP strategies can be reformulated in that case as follows ::

GASP strategy :

(1) $Query(q) \wedge Ax(q,l) \rightarrow Th(q,l)$

(2) $Query(q) \wedge Th(q,l_1) \wedge Ax(l_1,l_2) \rightarrow Th(q,l_2)$

GALP strategy :

(1) $Query(l) \wedge Ax(l,l_1) \rightarrow Th(l,l_1)$

(2) $Query(l) \wedge Ax(l,l_1) \rightarrow Query(l_1)$

(3) $Query(l) \wedge Ax(l,l_1) \wedge Th(l_1,l_2) \rightarrow Th(l,l_2)$

From these rules we can see that the graph of the relation Th(x,y) is the transitive closure of the graph of the relation Ax(x,y). That is not surprising since on one hand the sentence Ax(A,B) is repesented in the graph by the two nodes A and B, and by an edge from B to A. On the other hand Ax(A,B) is also a representation of the formula : $A \leftarrow B$. Then there is a one to one correspondance between the computation of the Ax(x,y) transitive closure and the computation of the theorems of the form : $A \leftarrow B$.

The interesting point is that GASP and GALP correspond to two quite different strategies to compute the transitive closure of a graph. In fact we does not have to compute the overall closure but only the edges arriving at the node representing the query.

In the GASP strategy only edges arriving at the query node are computed. In the GALP strategy all the edges arriving at the query node, or at one of its predecessors, are computed.

For example, if we have in T the clauses :

$Q \vee \neg A \quad A \vee \neg B \quad B \vee \neg C \quad C \vee \neg D$

·represented by :

$Ax(Q, A) \quad Ax(A, B) \quad Ax(B, C) \quad Ax(C, D)$

the GASP strategy generates Th(Q,B), from Th(Q,A) and Ax(A,B), while the GALP strategy generates Query(A) from the initial query and all the corresponding relevant theorems wich are : Th(A,B), Th(A,C) and Th(A,D). None of these sentences are computed in the GASP strategy.

The figure 1 shows the Ax(x,y) relation graph and the Th(x,y) relation graph edges computed by GASP. The figure 2 shows the edges computed by GALP.

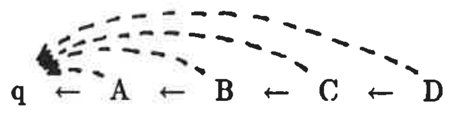

Figure 1: Theorems computed by GASP strategy

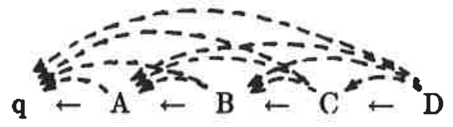

Figure 2: Theorems computed by GALP strategy

From this very simple example we can understand that GALP always generates more sentences than GASP. Let's consider now a theory T defining a graph for the relation Ax(x,y) which is a tree, where each node has exaclty k predecessors and the tree has n levels. In that case, when k is fixed and n grows, the number of edges computed by GASP is of the order of $2k^n$ while the number of edges computed by GALP is of the order of nk^n; the ratio is $n/2$, and is independent of k; moreover in most applications n is less than or equal to 5. These results give a quantitative idea of how much GALP is less efficient than GASP.

A natural conclusion would be to drop GALP strategy. However the conclusion is not so obvious if we consider the issue of theories containing recursive definitions, in the sense defined by Henshen and Naqvi in [9].

Let's consider the very standard example of the ancestors. If we have in T the clauses :

$L(x,y) \vee \neg \text{Ancestor}(x,y) \qquad \text{Ancestor}(x,y) \vee \neg \text{Father}(x,y)$
$\text{Ancestor}(x,y) \vee \neg \text{Ancestor}(x,z) \vee \neg \text{Father}(z,y)$

If we ask the query : L(x,y)? the conditional answer will contain an infinite number of clauses, even if subsumed clauses are eliminated. Indeed we will get all the formulas of the form :

$$L(x,y) \vee \neg \text{Ancestor}(x,z_i) \vee \neg \text{Father}(z_i,z_{i-1}) \vee \ldots \vee \neg \text{Father}(z_2,z_1) \vee \neg \text{Father}(z_1,y)$$

The problem of having infinite answers is independent of the strategy. There are several possible approach to solve the problem, the first one is to find a finite repesentation for this infinite set, the second one is to cut the answer computation according to some appropriate criterium, and the third one is to consider a more restrictive definition of conditional answers leading to finite answers.

We have investigated the third approach restricting conditional answers to extensional conditional answers. In that case, since we does not have functional symbol, the number of distinct ground literals is finite, and the number of ground clauses is also finite. Nevertheless it is not obvious to design a strategy generating ground clauses which does not generate also infinite sets of non ground clauses.

We shall see in the next section how the GALP strategy can be adapted for this purpose. The basic idea is to delay theorems generations until we are sure to be able to generate a "ground proof tree": i.e. a proof tree where all the clauses, except some axioms, are ground clauses.

That is the reason why the GALP strategy should not be droped without more investigations.

6 Strategy for Extensional Conditional Answers

The strategy adapted from GALP is first informally presented with examples.

Let's consider the theory T :

(1) $Sxy \lor \neg Pxy \lor \neg P'xy$ (2) $Pxy \lor \neg Qxz \lor \neg Rzy$ (3) $P'xy \lor \neg Q'xt \lor \neg R'ty$
(4) Qad (5) $R'cd$

If the query is : $Sxy?$ we can derive the ground answer (8) with the following proof :

(2) (4) $\Rightarrow$ (6) $Pay \lor \neg Rdy$
(3) (5) $\Rightarrow$ (7) $P'xb \lor \neg Q'xc$
(1) (6) (7) $\Rightarrow$ (8) $Sab \lor \neg Rdb \lor \neg Q'ac$

However this proof is not a ground proof as defined before, since (6) and (7) are not ground. Moreover we can easily check there is no possility to generate a ground proof tree because (4) (resp. (5)) can be resolved only with (2) (resp. (3)), and any resolvent from (1), (2) and (3) is not ground because the parent clauses does not contain constants.

That example shows that to adapt GALP we have in some particular cases to infer from a clause some particular instantiation of that clause. Remember that in any strategy based on Resolution Principle we cannot infer a clause only by instantiating some variables. For example we cannot directly infer : $Pab \lor \neg Qac \lor \neg Rcb$ from (2) $Pxy \lor \neg Qxz \lor \neg Rzy$.

Nevertheless if we want to be able to generate only ground proof trees we need this kind of inference if we want to generate only ground proof trees, as shows the previous example, but we have to carefully apply this kind of instantiation. We have to apply it only when we have a reason to support it. The reason adopted in the adapted version of GALP is to allow to instantiate an axiom with some constants only when we know that these constants will instantiate corresponding variables in further inferences in the proof.

The problem now is to define a method to discover which variables will be instantiated without generating a complete proof tree. For this purpose we analyse how resolved literals allow to propagate the constants.

This analysis can be desribed with the same example.

First we generate all the queries and subqueries. That is informally represented by :

Sxy? and (1) $\Rightarrow$ Pxy? and P'xy?
Pxy? and (2) $\Rightarrow$ Qxz? and Rzy?
P'xy? and (3) $\Rightarrow$ Q'xt? and R'ty?

The axiom (4) provides an answer to the query Qxz? which instantiates x with a, and z with d; that is represented by Qad $\uparrow$. In the same way Qad $\uparrow$ and the axiom (2) instantiate x with a in answers to Pxy?; that is represented by Pay $\uparrow$. Then informally we have :

Qxz? and(4) $\Rightarrow$ Qad $\uparrow$
Qad $\uparrow$ and (2) $\Rightarrow$ Pay $\uparrow$
R'ty? and (5) $\Rightarrow$ R'cb $\uparrow$
R'cb $\uparrow$ and (3) $\Rightarrow$ P'xb $\uparrow$

Now we notice that any clause containing Pay, or one of its instances, instantiates x with a in $\neg$P'xy when it is resolved with (1). Then, in every clause containing P'xy which is resolved with (1) and a clause containing Pay, the variable x can be instantiated with a *before* the resolution. In particular (2) can be instantiated with a/x before (2) is involved in the resolution with (1). The fact that x can be instantiated with a in P'xy is represented by P'ay $\downarrow$. Then informally we have :

Pay $\uparrow$ and (1) $\Rightarrow$ P'ay $\downarrow$
P'xb $\uparrow$ and (1) $\Rightarrow$ Pxb $\downarrow$
Pxb $\downarrow$ and (4) and (2) $\Rightarrow$ (6') Pab $\vee$ $\neg$Rda
P'ay $\downarrow$ and (5) and (3) $\Rightarrow$ (7') P'ab $\vee$ $\neg$Q'ac

Finally we have :

(1) (6') (7') $\Rightarrow$ (8) Sab $\vee$ $\neg$Rdb $\vee$ $\neg$Q'ac

The next example shows that when constants are propagated "down", we have to keep trace of the inference which allows to instantiate variables with these constants.

Let's consider a theory T with the clauses :

(1) $Px \vee \neg Qx \vee \neg Rx$ (2) $Px \vee \neg Sx \vee \neg Rx$ (3) $Rx \vee \neg Tx$ (4) Qa

If the initial query is Px?, with the same notations, we have :

Px? and (1) $\Rightarrow$ Qx? and Rx?
Px? and (2) $\Rightarrow$ Sx? and Rx?
Rx? and(3) $\Rightarrow$ Tx?
Qx? and (4) $\Rightarrow$ Qa $\uparrow$
Qa $\uparrow$ and (1) $\Rightarrow$ Ra $\downarrow$
Ra $\downarrow$ and (3) $\Rightarrow$ (3') $Ra \vee \neg Ta$
(3') and (4) and (1) $\Rightarrow$ (5) $Pa \vee \neg Ta$
(3') and (2) $\Rightarrow$ (6) $Pa \vee \neg Sa \vee \neg Ta$

In fact there is no reason supporting the derivation of (6) because the resolvent of (2) and (3) is (7) $Px \vee \neg Sx \vee \neg Tx$ which cannot be resolved with Qa. Then there is no proof allowing to instantiate x with a in (7). To avoid this truble we have to memorize that Ra $\downarrow$ is generated in the context of an inference involving (1) and Qa $\uparrow$, and we have to prevent the use of (3') in another inference context. If we denote the inference context in which constants can be propagated down by : (1,Qa $\uparrow$) we have :

Px? and (1) $\Rightarrow$ Qx? and Rx?
Px? and (2) $\Rightarrow$ Sx? and Rx?
Rx? and(3) $\Rightarrow$ Tx?
Qx? and (4) $\Rightarrow$ Qa $\uparrow$
Qa $\uparrow$ and (1) $\Rightarrow$ Ra $\downarrow$(1,Qa $\uparrow$)
Ra $\downarrow$(1,Qa $\uparrow$) and (3) $\Rightarrow$ (3') $Ra \vee \neg Ta$(1,Qa $\uparrow$)
(3') and (4) and (1) $\Rightarrow$ (5) $Pa \vee \neg Ta$

Where (3') $Ra \vee \neg Ta$(1,Qa $\uparrow$) means that the clause (3') can be involved only in resolutions of the type : (1,Qa $\uparrow$) . This information prevents (3') to be resolved with (2).

We consider that two literals l $\uparrow$ (resp. l $\downarrow$) and l' $\uparrow$ (resp. l' $\downarrow$) which differ only by the name of variables are *equivalent*. Therefore the process stops after a finite number of steps even if we have recursive definitions because only ground theorems are generated.

The adapted GALP version is called GRALP ("GR" is for ground). For its definition we have to add new notations to those introduced in section 4.

Notations :

- IQuery(l) : l is the initial query.
- Query(l) : l is a subquery.
- Up(l) : the constants in l are in the query, or are transmitted from the answers to some subqueries to the answer to the query; it was previously denoted by l $\uparrow$.
- Dni(l,(inf)) (read Down(l,(inference))) : the constants in l result of the unification of the other literals in the inference "inf", and can be used to instantiate axioms containing a literal more general than l.
- Dn(l) : same definition as Dni; the only difference is that the inference in which l is intantiated is not explicited; it was previously denoted by l $\downarrow$.
- (inf)=(α, Dn(l)/i_0, Up($\neg l_1$)/i_1, $\ldots$,Up($\neg l_p$)/i_p) : "inf" is an inference where α is the name of an axiom of the form : $l'_0 \vee l'_1 \vee \ldots \vee l'_n$, and l is unifiable with l'_{i_0}, l_1 is unifiable with l'_{i_1}, $\ldots$ l_p is unifiable with l'_{i_p}. To have simpler notations sometimes we shall assume : $i_0 = 0 \quad i_1 = 1 \ \ldots \ i_p = p$. That give the notation :

  (inf)=(α, Dn(l), Up($\neg l_1$), $\ldots$,Up($\neg l_p$))
- Th(c,(inf)) : c is a theorem which can be used only in an inference of type "inf".
- Comp(inf) : if Th(c,(inf)) is used in a particular inference inf' involving an axiom and some other theorems, Comp(inf) checks that this axiom and these theorems are compatible with the conditions expressed by inf; i.e. the axiom must be α, some of the resolved literals in the theorems involved in inf' must be instances of $\neg l_1, \ldots, \neg l_p$, and the inference inf' must be dependent on a query which is an instance of l.
- Ax(α : c) : c is an axiom called α.
- GAx(α : c) : c is a ground instance of the axiom α.

GRALP Definition

(1) $\mathrm{IQuery}(l) \rightarrow \mathrm{Dni}(l, (\emptyset))$

(2) $\mathrm{IQuery}(l) \rightarrow \mathrm{Query}(l)$

For each i in [1,n] :

(3.i) $\mathrm{Query}(l) \wedge \mathrm{Ax}(\alpha : l \vee l_1 \vee \ldots \vee l_i \vee \ldots \vee l_n) \rightarrow \mathrm{Query}(\neg l_i)$

Endfor;

(4) $\mathrm{Query}(l) \wedge \mathrm{Ax}(\alpha : l \vee c) \rightarrow \mathrm{Up}(l)$

(5) $\mathrm{Query}(l) \wedge \mathrm{Up}(\neg l_1) \wedge \ldots \wedge \mathrm{Up}(\neg l_n) \wedge \mathrm{Ax}(\alpha : l \vee l_1 \vee \ldots \vee l_n \vee c) \rightarrow \mathrm{Up}(l)$

For each j in [p+1,n] :

(6.i) $\mathrm{Dni}(l, (\mathrm{inf})) \wedge \mathrm{Up}(\neg l_1) \wedge \ldots \wedge \mathrm{Up}(\neg l_p) \wedge \mathrm{Ax}(\alpha : l \vee l_1 \vee \ldots \vee l_p \vee \ldots \vee l'_j \vee \ldots \vee l_n)$
$\rightarrow \mathrm{Dni}(\neg l'_j, (\alpha, \mathrm{Dn}(l), \mathrm{Up}(l_1), \ldots, \mathrm{Up}(l_n)))$

Endfor;

(7) $\mathrm{Dni}(l, (\mathrm{inf})) \wedge \mathrm{GAx}(\alpha : l \vee c) \rightarrow \mathrm{Th}(l \vee c, (\mathrm{inf}))$

(8) $\mathrm{Dni}(l, (\mathrm{inf})) \wedge \mathrm{GAx}(\alpha : l \vee l_1 \vee \ldots \vee l_n \vee c) \wedge \mathrm{Th}(\neg l_1 \vee c_1, (\mathrm{inf}_1)) \wedge \mathrm{Comp}(\mathrm{inf}_1) \wedge$
$\ldots \wedge \mathrm{Th}(\neg l_n \vee c_n, (\mathrm{inf}_n)) \wedge \mathrm{Comp}(\mathrm{inf}_n) \rightarrow \mathrm{Th}(l \vee c_1 \vee \ldots \vee c_n \vee c, (\mathrm{inf}))$

where, if we have $\mathrm{Comp}(\mathrm{inf}_i) = \mathrm{Comp}(\alpha : \mathrm{Dn}(l), \mathrm{Up}(\neg l_{i_1}), \ldots, \mathrm{Up}(\neg l_{i_q}))$ where for each i_j we have i_j in [1,n], $\mathrm{Comp}(\mathrm{inf}_i)$ is satisfied by a set of sentences S iff there is in S sentences unifiable with the following sentences :

$\mathrm{Dni}(l, (\mathrm{inf}))$ $\quad$ $\mathrm{Ax}(\alpha : l \vee l_1 \vee \ldots \vee l_n \vee c)$
$\mathrm{Th}(\neg l_i \vee c_i, (\alpha : \mathrm{Dn}(l), \mathrm{Up}(\neg l_{i_1}), \ldots, \mathrm{Up}(\neg l_{i_q})))$

and for each j in [1,q] :

$\mathrm{Th}(\neg l_{i_j} \vee c_{i_j}, (\mathrm{inf}_{i_j}))$

Comments on the GRALP Definition :

- Rules (1) and (2) : we have particular rules for the initial query because the constants in the initial query can be propagated down without any constraints about some particular inference.

- Rule (4) : if the literal l in the axiom $l \vee c$ contains some constants, these constants will be propagated if the axiom is resolved with another clause on the literal l. Example :

  From : Query(Pxy) and Ax(α : Paz $\vee \neg$Rz), the rule (4) allows to generate Up(Pay).

- Rule (5) : if clauses containing literals like $\neg l_i$ are resolved with the axiom, some constants in the $\neg l_i$s will be propagated into l. Example :

  From : Query(Sxy) and Ax(β : Sxy $\vee \neg$Pxz $\vee \neg$Qzy) and Up(Pay), the rule (5) allows to generate Up(Say).

- Rule (6.i) : constants which are propagated down by a subquery l, and constants which are propagated up by clauses containing $\neg l_i$, are transmitted to the other literals by the unification process and are propagated down via the literals l'_js. In that rule we may have p=0. Example :

  From : Dni(Sat,(inf)) and Ax(γ : Sxy $\vee \neg$Vxz $\vee \neg$Vzy) and Up(Vbu), the rule (6.i) allows to generate : Dni(Vab,(γ, Dn(Sat)/1, Up(Vbu)/3)) and Dni(Vaz,(γ, Dn(Sat)/1)). Here Up(Vbu)/3 imposes the condition that any inference of type (γ, Dn(Sat)/1, Up(Vbu)/3)) must involve a clause containing an instance of Vbu, for example the clause Vbc $\vee$ Tbc, and that this instance has to be resolved with third literal in γ, that is $\neg$Vzy.

- Rule (7) : the inference inf which has allowed to generate Dni(l,(inf)) is transmitted to the generated ground theorem $l \vee c$. Then this theorem can be only involved in inferences of type inf. Example :

  From : Dni(Vaz,(γ, Dn(Sat)/1)) and Ax(δ : Vxd $\vee \neg$Ux), the rule (7) allows to generate : Th(Vad $\vee \neg$Ua, (γ, Dn(Sat)/1)).

  The premise GAx(α : $l \vee c$) in the rule means that there must be in S a sentence Ax(α : $l' \vee c'$) such that, after unification with σ, $(l' \vee c')\sigma$ is a ground clause.

- Rule (8) : all the premises of the form Comp(inf$_i$) check that the clauses $\neg l_i \vee c_i$ are involved in an appropriate inference. Example :

  From : Dni(Sxy, $\emptyset$) Ax(α : Sxy $\vee \neg$pxy $\vee \neg$P'xy)
  Th(Pab $\vee \neg$Rda, (α, Dni(Sxy)/1, Up(P'xb)/3))
  Comp(α, Dni(Sxy)/1, Up(P'xb)/3)

Th($P'ab \vee \neg Q'ac$, (α, Dni(Sxy)/1, Up(Pay)/2)
Comp(α, Dni(Sxy)/1, Up(Pay)/2)
the rule (8) allows to infer Th($Sab \vee \neg Rdb \vee \neg Q'ac, \emptyset$).

The premise Comp(α, Dni(Sxy)/1, Up(P'xb)/3) means that the inference must involve a theorem where the resolved literal is an instance of P'xb, and this literal must be resolved upon the third literal P'xy in α.

- Inference type and inference instance : in general inf refers to an inference type. For example inf=(γ, Dn(Sat)/1, Up(Vbu)/3)) represents an inference type. An inference instance of this type is any inference where a γ instance obtained by unification of Sat and its first literal is resolved with a set of clauses such that one of them contains an instance of Vbu which is resolved with the third literal of γ. For example the inference involving the clauses :

  $Say \vee \neg Vaz \vee \neg Vzy \quad Vbc \vee Tbc \quad Vab \vee Tab$

  where the resolvent is : $Sac \vee Tab \vee Tbc$.

- Subsumed sentences : here we have to change the definition of subsumed sentences. The reason is that, if we have Up(Pabx) and Up(Payz) we don't have to remove Up(Pabx) because it carries an information not derivable from Up(Payz). However Up(Pabx) and Up(Pabz) carry the same information and we can remove one of them. It is the same for Dn(abx) and Dn(ayz).

  In the new definition we say Up(l) (resp. Dn(l)) subsumes Up(l') (resp. Dn(l')) iff l is equal to l', up to a variable renamimg. We say inf subsumes inf' iff inf is equal to inf', up to a variable renaming. We say Th(c,(inf)) subsumes Th(c'(inf')) iff inf subsumes inf' and c subsumes c' in the usual sense. We say IQuery(l) (resp. Query(l)) subsumes IQuery(l') (resp. Query(l')) iff l subsumes l' in the usual sense.

A least fixed point operator can be associated to GRALP like for GALP. It is easy to check that the least fixed point is computed after a finite number of steps. That is because in the generated sentences of the form Th(c,(inf)) c is always a ground clause, and because the literals which are arguments of the predicates Up, Dn, Dni, Query and IQuery are in a finite number (up to a variable renaming).

7 Conclusion

We have presented two strategies to generate the conditions which allow to know the answer to a query in the context of a non-Horn Deductive Data Base. The basic idea is to focus as far as possible the derivation process on clauses which are relevant for the query.

We have shown that the GASP strategy is always more efficient than the GALP strategy. A least fixpoint operator can be associated to both strategies. This computation technique prevents to repeat several times the computation of the answer to the same query or subquery. That is a significant advantage with respect to computation techniques "a la Prolog".

In the case of recursive definitions the answer may be infinite. However if we restrict the answer to ground clauses the answer is finite because we don't have functional symbols. For this particular case we have designed the GRALP strategy, an adaptation of GALP to produce only ground clauses. The associated least fixpoint operator always compute the answer in a finite number of steps. However at this time we have no result about the completeness of GRALP strategy because we have no formal definition of an extensional conditional answers. That needs more investigations, and the GRALP definition must be considered as a work in progress.

It should also be clear that the definition of these strategies has to be considered as a general framework for further refinements. Indeed there are many open choices to implement these strategies, and, depending on these choices, the performances can be strongly improved.

References

[1] F. Bancilhon and R.. Ramakrishnan. An amateur's introduction to recursive query processing strategies. In *Proc. ACM PODS*, 1986.

[2] L. Cholvy and R.. Demolombe. Querying a rule base. In *Proc. of 1st Int. Conf. on Expert Database Systems*, 1986.

[3] R.. Demolombe. An efficient evaluation strategy for Non-Horn Deductive Data Bases. In *IFIP Congress'89*. Elsevier, 1989.

[4] R.. Demolombe and L. Fari nas del Cerro. Two Inference Rule for Hypothesis Generation. Technical report, ONERA-CERT, 1990.

[5] E. Lozinskii. Evaluating queries in deductive databases by generating. In *Proc of IJCAI*, 1985.

[6] E. Lozinskii. Computing facts in non-horn deductive systems. In *Proc of VLDB*, 1988.

[7] J. Minker. On indefinite databases and the closed world assumption. In *Proc. of 6th Conference on Automated Reasoning*, 1982.

[8] J. Minker and A. Rajasekar. Procedural interpretation of non-horn logic programs. In *Proc. Conference on Automated Deduction*, 1988.

[9] S. Naqvi and L. Henshen. Recursive query answering with non-horn clauses. In *Proc. Conference on Automated Deduction*, 1988.

[10] R.. Reiter. Nonmonotonic reasoning. In *Annual Reviews of Computer Science, 2*, 1987.

[11] R.. Reiter and de Kleer. Foundations of assumption-based truth maintenance system. In *AAAI-87*, 1987.

[12] 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.

[13] 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.