A Strategy for the computation of Conditional Answers

Robert Demolombe*

CERT-ONERA Département d'Etudes et Recherches en Informatique
2 Avenue E.Belin B.P. 4025, 31055 Toulouse, France

Abstract.

We consider non-Horn Deductive Data Bases (DDB) represented in a First Order language without function symbols. In this context the DDB is an incomplete description of the world. A first approach to reduce the incompleteness is to add to the DDB some kind of default rules, in order to automatically assume missing information. The second approach, which is adopted in this paper, is to provide to the user the conditions which 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 derive the more general conditions c such that : $T \vdash q \leftarrow c$. We present a strategy, GASP, designed for this purpose. It is defined by meta rules, and these meta rules can be used for a least fixpoint operator definition. We show that the GASP strategy is always more efficient than another usual strategy called GALP. Since in the case of recursive definitions the answers may be infinite GASP has been adapted into GRASP in order to only compute ground conditional answers. We show that the least fixpoint operator associated to GRASP computes the answer in a finite number of steps, even if the DDB contains recursive definitions.

1 Introduction

Many works have been devoted to the *standard* approach of Deductive Data Bases (DDB) [1, 11, 5, 12]. 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, 2, 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 a new kind of answers called *Conditional Answers*. We consider two kinds of Conditional Answers : the Intensional Conditional Answers, that are derived from IDB, and the Extensional Conditional Answers, that are derived from IDB $\cup$ EDB. Conditional answers are another way to deal with incompleteness. Indeed the usual approach is to reduce the incompleteness with some kind of meta rule like Closed World Assumption (CWA), or Generalised Closed World Assumption (GCWA) [7], in the

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

context of disjunctive DDB, or default rules in the context of non-monotonic reasoning [9]. 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 provides the less restrictive assumptions that allow to infer the query.

Let's consider for example the very simple DDB : $A \vee B \leftarrow C \wedge D$, C, and the query : A?.

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

In the next section is presented a general definition of conditional answers. Then we present a strategy to compute conditional answers. Its efficiency is compared with another standard strategy, and we point out the particular problem of infinite answers. In the last section we propose a modification to this strategy in order to compute extensional conditional answers in a finite number of steps.

2 General definition of Conditional Answers

We consider queries that 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 :
$T = IDB \cup EDB \cup Q$, where all the formulas are represented in clausal form, and each clause is Range Restricted. Moreover, as usual in the DDB context, we consider clauses **without functional symbols**.

Definition 1 : Conditional Answer

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 d is minimal with regard to subsumption, in the context of a conditional answer, if there is no clause d' in that conditional answer such that d' subsumes d. A clause d' subsumes a clause d if there exists a substitution σ such that : $d'\sigma \subseteq d$.

The clause c is called by Reiter and de Kleer, in [10], a minimal support for $q\sigma$.

It is important to notice that computing Conditional Answers is a new kind of problem with regard to Theorem Proving and Logic Programming. The new feature comes from the fact that an answer is neither a truth value, like in Theorem Proving, nor a set of substitutions, like in Logic Programming, but a set of clauses.

This problem is deeply related to Abductive Reasoning, with some particular features due to the DDB context.

Definition 2 : Extended Conditional Answer

With the same notations we define an extended conditional answer as 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 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; this means that $\neg c$ may be an inconsistent assumption. In this paper we do not consider the problem of checking the consistency of $\neg c$.

3 Intuition of the strategy

The GASP strategy presented in this section is based on the L-inference presented in [3]. The strategy is called GASP, an abreviation for Generate As Soon as Possible. It is informally described in this section with a simple example. For this decription we shall call *relevant theorem* for a given query, a clause derivable from T containing the query q, or one of its instances.

The idea is, in a first step, to select the axioms in T which are relevant theorems.In the current step one, or several, generated relevant theorems are resolved with an axiom in T. The resolvent is a new relevant theorem which can be used in the next step. At each step tautologies and subsumed clauses are removed.

Let's consider, for example, the theory T with the axioms :

(1) $Px \vee \neg Qx$ (2) $Px \vee \neg Rx$ (3) $Qx \vee Rx \vee \neg Sx$
(4) $Ra \vee \neg Ta$

and the query [1] : Px?. The clauses generated by the GASP strategy are :

Step 1 : (1) $Px \vee \neg Qx$ (2) $Px \vee \neg Rx$
Step 2 : (5) $[Px \vee Rx \vee \neg Sx]$
(6) $[Px \vee Qx \vee \neg Sx]$ (7) $Px \vee \neg Sx$ (8) $Pa \vee \neg Ta$

In the Step 1 are generated the axioms in T containing an instance of the query Px. In the Step 2 a standard resolution generates (5) (resp. (6)) from (1) (resp. (2)) and (3). An hyperresolution generates (7) from (1), (2) and (3), and (8) is generated from (2) and (4). At the end of Step 2 the clauses (5) and (6) are removed because they are subsumed by (7). Notice these resolutions preserve Px, or an instance of Px, in the resolvent.

As we said before for each clause in the extended conditional answer it would be possible to check if the condition is consistent or not with T in a further phase. For example to check if $\neg Ta$ is a theorem, or not, we could apply again GASP to the query : $\neg Ta$?.

[1] The predicate arguments are not between parenthesis to have simpler notations. For example, P(x,a) is noted Pxa.

4 Formal definition of the strategy

In this section the strategy is formally defined by meta rules. These rules express, at a meta level, the derivation control. 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. We use the following notations for meta-predicates and meta-variables :

Query(l) : the extended conditional answer to the query l has to be computed.
Ax(c) : c is an axiom of T.
Th(c) : c is a theorem of T.

q, l_i : denote literals at the object level.
c_i : denotes a clause variables.
$l \vee l_i \vee c_i$: denotes the set of literals : $\{l\} \cup \{l_i\} \cup c_i$.

Definition 3 : GASP Strategy

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

(2) $Query(q) \wedge Th(q \vee l_1 \vee c_1) \wedge \ldots \wedge Th(q \vee l_n \vee c_n) \wedge Ax(\neg l_1 \vee \ldots \neg l_n \vee c_0) \rightarrow$
$Th(q \vee c_1 \vee \ldots \vee c_n \vee c_0)$

We have used dots here to make the definition more easy to read. It would not be difficult to replace the dots by recursive definitions.

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 meta rules are evaluated by the trivial strategy : incremental saturation by level, with elimination of subsumed clauses and tautologies. Here incremental means that when a new sentence is generated by a meta rule , at least one of its premisses in the meta rule has to be a new sentence in the computation of the previous level. The sets of sentences generated by saturation by level are denoted by : $S_0, \ S_1, \ \ldots \ , S_i, \ \ldots$.

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, or S contains sentences whose some factors are : Query(Q), $Th(Q_1 \vee L_1 \vee C_1), \ \ldots, Th(Q_n \vee L_n \vee C_n), Ax(\neg L'_1 \vee \ldots \vee \neg L'_n \vee C_0)$; where Q, Q_1, L_1, C_1, $\ldots, Q_n$, L_n, C_n, C_0 are literals or clauses at the object level,
- there exists a most 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 : $q = Q\sigma \qquad l_i = L_i\sigma \qquad \neg l_i = \neg L'_i\sigma \quad c_i = C_i\sigma$, 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 $P(t'_1, \ldots, t'_p)$, are short hands for the set of equations : $t_1 = t'_1 \qquad t_2 = t'_2 \quad \ldots \quad t_p = t'_p$.

It is easy to show that the interpretation we have defined for the meta rules defining GASP provides a definition for a least fixpoint operator.

5 Comparison with other strategies

It is interesting to compare GASP with another very intuitive strategy based on the idea of the decomposition of problems into sub-problems " à la Prolog". Here the problem is to compute the extended conditional answer to a query : A?. If there is an axiom containing A in the theory T of the form : $A \vee \neg B_1 \vee \ldots \vee \neg B_i \vee \ldots \vee \neg B_n$, where the B_is may be positive or negative literals, we can generate sub-problems, i.e. new queries, of the form : $B_1?$, $B_2?, \ldots, B_i?, \ldots, B_n?$. Indeed we know that any answer to a query like $B_i?$ is of the from : $B_i \vee c_i$, and therefore any set of answers can be resolved by an hyperresolution with the axiom to generate new answers of the form : $A \vee c_1 \vee \ldots \vee c_i \vee \neg B_{i+1} \vee \ldots \vee \neg B_n$. This strategy is called GALP, which is an abreviation for Generate As Late as Possible. It can be defined by meta rules in the same style as for GASP.

Definition 4 : GALP Strategy

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

For each i in [1,p] :
(2.i) $\mathrm{Query}(l) \wedge \mathrm{Ax}(l \vee \neg l_1 \vee \ldots \vee \neg l_i \vee \ldots \vee \neg l_p) \rightarrow \mathrm{Query}(l_i)$

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

In the particular case where we impose the l_is to be positive literals, and c_0, $c_1, \ldots, c_n$ denote the empty clause, the GALP strategy is very close to strategies like : Magic Sets [1], ALEXANDER [11], QSQ [12], or APEX [5].

Unfortunately it can be shown that in every cases GALP generates a superset of the clauses generated by GASP, and then it is always less efficient. However, as it is noticed in [3], GASP may generate an infinite set of clauses when the initial theory contains recursive definitions. Nevertheless if we are interested in Extensional Conditional Answers containing only ground clauses, it is possible to adapt GASP in order to prevent infinite derivations. That is the purpose of the next section.

6 Strategy for Extensional Conditional Answers

The adapted strategy is based on the following interesting property of Range Restricted clauses : if a ground clause is derivable, by Resolution, from a set of Range Restricted clauses, and if s is the composition of all the most general unifiers used in the proof of that clause, then, if we apply s to any clause in the proof, we get a ground clause. The idea is to design a strategy, based on this property, which generates only "ground proof trees", i.e. proof trees where all the clauses are ground clauses. For this purpose we have to search, before to generate a theorem, if all its variables will be instantiated in further resolutions. If that it the case, a ground instance of the theorem corresponding to these instantiations is generated.

To find these instantiations we consider the connection graph (as defined by Henschen and Naqvi in [4]) of the set of axioms. In this graph the nodes are clauses, and we can imagine these clauses as active agents able to send queries, or to receive answers, along the edges of the graph, and able to store the answers. These queries and answers are respectively represented in the meta rules by l? and l!. Their role is to find the unifiers of the proofs whose composition can lead to ground unifiers. They are of a different sort than the initial query and the conditional answers.

The strategy is called : GRASP, for "Generate gRound As Soon As Possible". Its description in terms of meta-rules is presented in the Definition 5.

Definition 5 : GRASP Strategy

(1) $\mathrm{Query(q)} \rightarrow \mathrm{q?}$

For each i in [1,n] :
(2.i) $\mathrm{l?} \wedge \mathrm{Ax}(\mathrm{l} \vee \mathrm{l}_1 \vee \ldots \vee \mathrm{l}_i \vee \ldots \vee \mathrm{l}_n) \rightarrow \neg \mathrm{l}_i?$

(3) $\mathrm{l?} \wedge \neg \mathrm{l}_{j_1}! \wedge \ldots \neg \mathrm{l}_{j_p}! \wedge \mathrm{Ax}(\mathrm{l} \vee \mathrm{l}_1 \vee \ldots \vee \mathrm{l}_n) \rightarrow \mathrm{l!}$

(4) $\mathrm{Query(q)} \wedge \neg \mathrm{l}_{j_1}! \wedge \ldots \neg \mathrm{l}_{j_p}! \wedge \mathrm{GrAx}(\mathrm{q} \vee \mathrm{l}_1 \vee \ldots \vee \mathrm{l}_n) \rightarrow \mathrm{Th}(\mathrm{q} \vee \mathrm{l}_1 \vee \ldots \vee \mathrm{l}_n)$

(5) $\mathrm{Query(q)} \wedge \mathrm{l}_{j_1}! \wedge \ldots \wedge \mathrm{l}_{j_p}! \wedge \mathrm{Th}(\mathrm{q} \vee \mathrm{l}_1 \vee \mathrm{c}_1) \wedge \ldots \wedge \mathrm{Th}(\mathrm{q} \vee \mathrm{l}_n \vee \mathrm{c}_n) \wedge \mathrm{GrAx}(\neg \mathrm{l}_1 \vee \ldots \neg \mathrm{l}_n \vee \mathrm{c}_0) \rightarrow$
$\mathrm{Th}(\mathrm{q} \vee \mathrm{c}_1 \vee \ldots \vee \mathrm{c}_n \vee \mathrm{c}_0)$

where i, $j_1, \ldots, j_p$ are in [1,n].

Rules of type (2.i) produce subqueries. The rule (3) allows to know, with the answer l!, the possible instantiations if the resolved literals in the axiom are $\neg \mathrm{l}_{j_1}! \wedge \ldots \neg \mathrm{l}_{j_p}!$, and if the query is l. In the rules (4) and (5), GrAx(c) means that c is a ground instance of an axiom. Rule (4) shows that only relevant ground theorems are generated, and the rule (5) combines these theorems to generate new relevant ground theorems.

The least fixpoint computation defined by these rules always terminates because there is no function symbol in the language, and the number of distinct l? and l! is finite. For the same reason the number of generated theorems is finite.

7 Conclusion

We have presented a strategy (GASP) to generate conditional answers in the context of a non-Horn Deductive Data Base. The basic idea is to focus the derivation process on clauses which are relevant for the query.

We have compared this strategy (GASP) with an another strategy (GALP) similar to standard strategies used in the context of Deductive Data Bases for Horn clauses, and we have shown that GASP is always more efficient than GALP. GASP is defined by meta rules, and a least fixpoint operator can be associated to these rules. This computation technique prevents to repeat several times the same computation. This is a significant benefit with respect to computation techniques "à la Prolog", or based on SL-resolution.

In the case of recursive definitions the answer may be infinite. For this particular case we have designed the GRASP strategy, an adaptation of GASP in order to derive 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 GRASP, because we have no denotational definition of what is computed by GRASP. That needs more investigations, and as to be considered as a work in progress.

It should also be clear that the definitions of these strategies have 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.

Acknowledgements : Many thanks to Luis Fariñas del Cerro for all our fruitfull and stimulating discussions.

References

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

[2] R. Demolombe. An efficient evaluation strategy for Non-Horn Deductive Data Bases. In *IFIP Congress'89, extended version in : Journal of Theoretical Computer Science Num 78*. Elsevier, 1989.

[3] R. Demolombe and L. Fariñas del Cerro. An Inference Rule for Hypothesis Generation. In *Proc. of International Joint Conference on Artificial Intelligence*, Sydney, 1991.

[4] L.J. Henschen and S.A. Naqvi. On Compiling Queries in Recursive First-Order Databases. *Journal of ACM*, 31,1, 1984.

[5] E. Lozinski. Evaluating queries in deductive databases by generating. In *Proc of 11th International Joint Conference on Artificial Intelligence*, 1985.

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

[7] J. Minker. On Indefinite Databases and the Closed World Assumption. In *Lecture Notes in Computer Science 138*. Springer Verlag, 1982.

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

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

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

[11] 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.

[12] 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.