

An Inference Rule for Hypothesis Generation

*

Robert Demolombe(1) Luis Fariñas del Cerro(2)

(1)ONERA/CERT

2 avenue E.Belin B.P. 4025

31055 Toulouse

France

e-mail : demolombe@tls-cs.cert.fr

(2)IRIT

Université Paul Sabatier

118 route de Narbonne

31052 Toulouse

France

e-mail : farinas@irit.fr

October 1990

Abstract

There are many new application fields for automated deduction where we have to apply abductive reasoning. In these applications we have to generate consequences of a given theory having some appropriate properties. In particular we consider the case where we have to generate the clauses containing instances of a given literal L . The negation of the other literals in such clauses are hypothesis allowing to derive L .

In this paper we present an inference rule, called L -inference, which was designed in order to derive those clauses, and a L -strategy. The L -inference rule is a sort of Input Hyperresolution. The main result of the paper is the proof of the soundness and completeness of the L -inference rule. The L -strategy associated to the L -inference rule, is a saturation by level with deletion of the tautologies and of the subsumed clauses. We show that the L -strategy is also complete.

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

1 Introduction

Traditionally automated deduction systems are devoted to prove if a given formula is a theorem; their applications, as is well known, have been very successful in many domains of Computer Science. Gradually this traditional functionality has been extended.

For example in Logic Programming, or in Deductive Databases, it is not enough to know if a closed formula is a theorem, indeed we want to know the set of substitutions used in the proof of a formula.

Recently applications, like ATMS, automated diagnosis, generation of “why not” explanations, conditional answering in Deductive Databases, partial deduction, all of them involving some kind of abductive reasoning, have emphasized the need of new functionalities. For every of these new applications it is necessary to produce, for a given formula, the set of hypothesis we have to add to a given theory to prove this formula. This shows that we are now expecting from an automated deduction method more information than an answer of the form : “yes”, or “no”, or a set of substitutions.

For these new applications (see [4]) the expected information is, for a given Database DB, and a given query Q which is not derivable from DB, the set of hypothesis X such that $X \rightarrow Q$ is derivable from DB, and X is as general as possible. Such X are called the *minimal supports* for Q by Reiter and de Kleer in [9].

In order to mechanize the production of hypothesis some new algorithms have been defined. For example, in the frame of Propositional Calculus, by Siegel [8], Cayrol and Tairac [6], Oxusoff and Rauzy [7], and Kean and Tsiknis [3], and in the frame of First Order Calculus, by Cholvy [1].

The objective of this paper is to present a new inference rule, and its associated strategy, which have been designed in order to efficiently compute the minimal support for a query.

We shall assume the reader is familiar with traditional theorem proving techniques as they are presented in [2].

2 General definition of the generation hypothesis problem

In all the paper we consider a First Order Language where formulas are in clausal form.

Let S be a set of clauses, a query adressed to S is represented by a literal L .

It is not restrictive to have only query represented by a single literal. Indeed, if the query is a first order formula F we can introduce a new atomic formula Q , and we can add to S the clauses generated by the clausal form of $F : F \rightarrow Q$. Then the answer to the query Q provides the minimal support for Q .

The answer to the query L , relative to S , is a set of clauses containing instances of L , defined by :

$$\text{ans}(L,S) = \{ L' \vee X \mid S \vdash L' \vee X, \text{ where } L' \text{ is an instance of } L, \text{ and } L' \vee X \text{ is not a tautology, and there is no clause } c \text{ derivable from } S \text{ such that } c \text{ subsumes } L' \vee X \}$$

According to this definition the clauses in $\text{ans}(L,S)$ are minimal with regard to the subsumption.

If $L' \vee X$ is in $\text{ans}(L,S)$, the negation of X is an hypothesis which allows to infer L' in the context of S .

It is worth noting some consequences of the condition of minimality. If $L' \vee X$ is minimal we have :

- L' is not derivable from S . That means we cannot give an answer to the query L' without additional hypothesis.
- X is not derivable from S . Therefore the corresponding hypothesis to infer L' is consistent with S .
- there exists no clause $L' \vee X'$ derivable from S such that $L' \vee X'$ subsumes $L' \vee X$. Since the condition $L' \vee X'$ subsumes $L' \vee X$ implies X' subsumes X , there is no weaker hypothesis than X to infer L' in the context of S .

A consequence of $L' \vee X$ not being a tautology is that X is not the trivial hypothesis $\neg L'$.

Example1 :

$$S = \{P(x, y) \vee \neg Q(x, y) \vee S(x, y), Q(a, b), \neg S(a, c), Q(b, d)\}$$

Let $L=P(a,x)$ be the query, the answer is :

$$\text{ans}(L,S) = \{P(a, x) \vee \neg Q(a, x) \vee S(a, x), P(a, b) \vee S(a, b), P(a, c) \vee \neg Q(a, c)\}$$

3 Definition of the Inference Rule

Definition 1. Let L be a literal. A clause C is an **L-clause** iff there is a literal L' in C such that L is an instance of L' .

Definition 2. Let L be a literal and let $M_i \vee e_i, 1 \leq i \leq p$, be a set E of clauses, called electrons, such that each M_i is a literal, and each e_i is an L -clause. Let n be the clause : $N_1 \vee N_2 \vee \dots \vee N_p \vee n'$, where the N_i 's are literals, which is called the nucleus. A finite sequence of L -clauses R_0, \dots, R_p is an **L-inference** iff :

- R_0 is n ,
- each R_{i+1} is a resolvent obtained (by the Resolution principle) from R_i and $M_i \vee e_i$, where the literal M_i is resolved against some instance of a literal N_j in $R_i, 1 \leq j \leq p$.

R_p is called the **L-resolvent** of E and n , and the R_i , for $1 \leq i \leq p - 1$, are called the **intermediate resolvents**. The L -inference will be represented by :

$$\frac{E \quad n}{R_p} .$$

Definition 3. Let S be a set of clauses. A **L-deduction** of C_n from S is a finite sequence $C_0 \dots C_n$ of clauses such that : each C_i is either a clause in S or there are C_{i_1}, \dots, C_{i_k} in the L -deduction, with each $i_j < i$, such that C_i is the L -resolvent of C_{i_1}, \dots, C_{i_k} , where the nucleus is in S .

Definition 4. A **R-deduction** of C_n from S is a finite sequence $C_0 \dots C_n$ of clauses such that : each C_i is either in S or there are C_{i_1}, C_{i_2} in the R -deduction, with each $i_j < i$, such that C_i is the resolvent (by the Resolution Principle) of C_{i_1} and C_{i_2} .

Theorem 1. (R.C.T. Lee [5]) Let S be a set of clauses, if c is a clause derivable from S , there is a clause c' , subsuming c , such that c' is derivable from S by a R-deduction.

Theorem 2. Let S be a set of clauses and L a given literal. If there is a R-deduction of $L \vee C$, then there is a L-deduction of $L \vee C$.

Proof. The proof is by induction on the number n of inferences in the R-deduction of $L \vee C$ from S .

For $n=1$, $L \vee C$ is the resolvent of two clauses C_1 and C_2 in S . Then either C_1 or C_2 (say C_1) is of the form $N \vee C'$, where C' is a L-clause, and N is the resolved literal. Therefore the R-inference is a L-inference.

For the induction step we assume we have a R-deduction of $L \vee C$ from S using n inferences. Let's consider in this R-deduction some clause C_0 which is the resolvent of two clauses C_1 and C_2 which are in S .

Then there is a R-deduction of $L \vee C$ from S and C_0 using $n-1$ inferences.

We distinguish the following two cases:

Case 1. C_0 is a L-clause. For the same reason as in the base induction step $\frac{C_1 \ C_2}{C_0}$ is an L-inference. Then by induction hypothesis is the R-deduction of $L \vee C$ from S and C_0 there exists a L-deduction D of $L \vee C$ from S and C_0 . If we add the L-inference : $\frac{C_1 \ C_2}{C_0}$ before the first occurrence of C_0 in D , we get a L-deduction of $L \vee C$ from S .

Case 2. C_0 is not a L-clause. Then by induction hypothesis there exists a L-deduction D of $L \vee C$ from S and C_0 . Assume the L-inference using C_0 , in D , is of the form $\frac{E \ C_0}{C_3}$, where E is a set of electrons and C_0 is the nucleus. We define a partition of the set E into two sets E_1 and E_2 as follows : a clause e in E is in E_1 if and only if the literal in e which is resolved in the inference $\frac{E \ C_0}{C_3}$ is provided by the clause C_1 .

Then we transform the proof $d : \frac{E \ \frac{C_1 \ C_2}{C_0}}{C_3}$,
 where $\frac{E \ C_0}{C_3}$ is an L-inference, and $\frac{C_1 \ C_2}{C_0}$ is an R-inference,
 into $d' : \frac{\frac{E_1 \ C_1}{C_4} \ \frac{E_2 \ C_2}{C_5}}{C_6}$.

The inferences $\frac{E_1 \ C_1}{C_4}$ and $\frac{E_2 \ C_2}{C_5}$ are L-inferences, because $\frac{E \ C_0}{C_3}$ is a L-inference, i.e. each intermediate resolvent in the L-inference is an L-clause, and the resolved literals in E_1 (resp. E_2) and C_1 (resp. C_2) are the same literals as the literals resolved in E and C_0 . Then C_4 and C_5 are L-clauses, and in the inference : $\frac{C_4 \ C_5}{C_6}$, the resolved literals are the same as in the inference : $\frac{C_1 \ C_2}{C_0}$; moreover these literals are not the literals which make C_4 and C_5 be L-clauses, then $\frac{C_4 \ C_5}{C_6}$ is an L-inference. So C_6 is an L-clause, and the transformed proof d' is an L-deduction.

The set of literals which must be unified, and resolved, in the proof d and in the proof d' are the same, then the two sets of equations defining the most general unifiers in d and d' are the same. Then the substitution applied to the literals of E , C_1 and C_2 , whose instances are in C_3 , is the same as the substitution applied to the literals of E_1 , E_2 , C_1 and C_2 whose instances are in C_6 . Therefore C_6 is identical to C_3 . Finally if we replace in D the proof d by d' we get an L-deduction of $L \vee C$ from S .

In the case where there are several occurrences of C_0 in the proof D , and C_0 is involved in several L-inferences of the form $\frac{E' \ C_0}{C'_3}$, we remove each C_0 occurrence with a similar transformation. Notice that each transformation decreases by one the number of C_0 occurrences. Then after a finite number of transformation of the L-deduction D we obtain a L-deduction of $L \vee C$ from S . Q.E.D.

The following completeness theorem is a trivial consequence of Theorem 1 and Theorem 2.

Theorem 3. Let S be a set of clauses, if $L \vee c$ is a clause derivable from S , there exists a clause $L' \vee c'$, subsuming $L \vee c$, which is derivable from S by an L-deduction.

Proposition Since every L-inference is a sequence of application of Resolution principle, the L-inference rule is sound.

One could notice that an L-deduction is very close to an OL-deduction, or an SL-deduction (see [5]) with top clause $\neg L$. However OL-resolution has been proved to be complete to generate the empty clause, but, as far as we know, there is no proof that OL-resolution is complete to generate clauses. The complexity of the proof of Theorem 1 clearly suggests that there is no evidence that a strategy which is complete to generate the empty clause is also complete for the generation of clauses. That is why the proof of Theorems 2 and 3 is, in our view, the main contribution of our work.

4 Definition of the L-strategy

A saturation algorithm is considered in order to define an effective procedure to compute the L-clauses. This algorithm could be improved using more sophisticated strategies like ordering.

The L-strategy computes the answer to a query into two steps. In the first step is computed a set of clauses, called extended answer, containing more clauses than the clauses in the answer. In the second step we have to remove all the clauses $L' \vee X$ in the extended answer such that X is derivable from S . In this second step the clause X is known, then testing if X is derivable from S can be done by any standard theorem proving strategy. For this reason this second step is not described in this paper, and we shall present only the first step.

Definition 5. We call **extended answer** the following set of clauses :

$$\text{eans}(L,S) = \{L' \vee X \mid S \vdash L' \vee X, \text{ where } L' \text{ is an instance of } L, \text{ and } L' \vee X \text{ is not a tautology, and there is no clause } c \text{ in } \text{eans}(L,S) \text{ such that } c \text{ subsumes } L' \vee X \}$$

Note that this definition is weaker than the previous one. Indeed, if $L' \vee X \in \text{ans}(L,S)$, X is not derivable from S , while if $L' \vee X \in \text{eans}(L,S)$, X may be derivable from S .

Notation : We denote $[S]$ a set of clauses where all the subsumed clauses have been removed.

Definition 6. Let S be a set of clauses, and let L be a query, the **L-strategy** computes the sets $S_0, \dots, S_{i+1}, \dots$, inductively as follows :

$$S_0 = [\{(L' \vee X)\sigma \mid L' \vee X \in S \text{ and } \sigma \text{ is the most general unifier of } L \text{ and } L' \}]$$

...

$$S_{i+1} = [S_i \cup \{L' \vee X \mid \text{there exists a set of electrons } E \text{ in } S_i, \text{ and a nucleus } n \text{ in } \mathbf{S}, \text{ such that } L' \vee X \text{ is the } L'\text{-resolvent of } E \text{ and } n\}]$$

...

For the purpose of the next definition we consider deductions as proof-trees instead of proof-sequences.

Definition We call **depth of a deduction** the number of inference steps in the longest path of the proof-tree corresponding to this deduction.

Theorem 4. If $L' \vee X$ is in $\text{eans}(L,S)$, where L' is an instance of L , then there exists some i such that $L' \vee X$ is in S_i .

Proof : The proof is by induction on the depth j of the L' -deduction of $L' \vee X$ from S .

For the base case ($j=0$) the proof is trivial.

For the induction step, assume there is an L' -deduction, of depth $j + 1$, of $L' \vee X$ from S , where the last L' -inference is of the form $\frac{E \quad n}{L' \vee X}$. Let $e = M \vee e'$ be an electron in E , where M is the resolved literal.

If e does not belong to $\text{eans}(L,S)$, there is an L -clause c in $\text{eans}(L,S)$ subsuming e . We distinguish the following two cases :

Case1 : c subsumes e' . Then c subsumes the instance of e' which is in $L' \vee X$. That contradicts the fact that $L' \vee X$ is in $\text{eans}(L,S)$.

Case2 : c does not subsume e' . Then c is of the form $M' \vee c'$, where M' is an instance of M . In this case we can transform the inference $\frac{E \quad n}{L' \vee X}$ by replacing e in E by c . Then the new L' -resolvent subsumes $L' \vee X$ because c' subsumes e' . That contradicts the fact that $L' \vee X$ is in $\text{eans}(L,S)$.

Therefore e belongs to $\text{eans}(L,S)$. Since the depth of the L' -deduction is equal to j , by induction hypothesis e is in S_j .

The same conclusion holds for any electron in E , so from the definition of S_{j+1} in function of S_j we can conclude that $L' \vee X$ is in S_{j+1} .

5 Some typical examples

In this section we present two examples showing the main features of our approach.

Example2 is a very simple example illustrating the interest of the L -strategy for automated diagnosis.

Let's consider a very simple system l , with components : b , b_1 , b_2 , and c , whose correct working is defined by the following rules and facts :

$l - \text{works} \leftarrow b - \text{works} \wedge c - \text{works}$
 $b - \text{works} \leftarrow b_1 - \text{works} \wedge b_2 - \text{works}$
 $b_1 - \text{works}$

If we respectively denote by : L , B , B_1 , B_2 and C , the propositions : l -works, b -works, b_1 -works, b_2 -works and c -works, we have :

$$S = \{ L \vee \neg B \vee \neg C, B \vee \neg B_1 \vee \neg B_2, B_1 \}$$

Query : L

$$S_0 = \{ L \vee \neg B \vee \neg C \}$$

$$S_1 = \{ L \vee \neg B \vee \neg C, L \vee \neg B_1 \vee \neg B_2 \vee \neg C \}$$

$$S_2 = \{ L \vee \neg B \vee \neg C, L \vee \neg B_2 \vee \neg C \}$$

The answer : $L \vee \neg B_2 \vee \neg C$ can be interpreted as : “ l -works if b_2 -works and c -works”, or as well as : “ a possible explanation that l does not work is that b_2 or c does not work”.

Example3, which is in Propositional Calculus, is interesting because it shows that the standard Input resolution strategy is not complete. Indeed with this strategy we cannot infer $L \vee A$, while $L \vee A$ is derivable with the L-strategy. Here we can see that the reason why the L-strategy is complete, although it is an Input strategy, is that the L-strategy is also an Hyperresolution.

$$S = \{ L \vee M \vee N, \neg M \vee N, M \vee \neg N, \neg M \vee \neg N \vee A \}$$

Query : L .

$$S_0 = \{ L \vee M \vee N \}$$

$$S_1 = \{ L \vee M, L \vee N \}$$

The clauses : $L \vee N \vee \neg N \vee A$, and $L \vee M \vee \neg M \vee A$ are not in S_1 because they are tautologies, and $L \vee M \vee N$ is not in S_1 because it is subsumed by $L \vee M$.

$$S_2 = \{ L \vee M, L \vee N, L \vee A \}$$

$L \vee A$ is generated by an L-inference where the two electrons are : $L \vee M$ and $L \vee N$, and the nucleus is : $\neg M \vee \neg N \vee A$. The clauses $L \vee \neg M \vee A$ and $L \vee \neg N \vee A$ are not in S_2 because they are subsumed by $L \vee A$.

Example4 shows how we can get infinite answers. The intuitive meaning of the query is : *What conditions implies that x is an ancestor of y ?*. Since the query does not refer to a specific set of persons, there is an infinite set of conditions which guarantee that x is an ancestor of y; each condition corresponds to a different level in the ancestor's hierarchy.

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

$$\text{Query} : L(x, y).$$

$$S_0 = \{L(x, y) \vee \neg \text{Ancestor}(x, y) \}$$

$$S_1 = \{L(x, y) \vee \neg \text{Ancestor}(x, y), L(x, y) \vee \neg \text{Father}(x, y), \\ L(x, y) \vee \neg \text{Ancestor}(x, z_1) \vee \neg \text{Father}(z_1, y) \}$$

$$S_2 = \{L(x, y) \vee \neg \text{Ancestor}(x, y), L(x, y) \vee \neg \text{Father}(x, y), \\ L(x, y) \vee \neg \text{Father}(x, z_1) \vee \neg \text{Father}(z_1, y), \\ L(x, y) \vee \neg \text{Ancestor}(x, z_2) \vee \neg \text{Father}(z_2, z_1) \vee \neg \text{Father}(z_1, y)\}$$

...

$$S_i = \{L(x, y) \vee \neg \text{Ancestor}(x, y), L(x, y) \vee \neg \text{Father}(x, y), \\ L(x, y) \vee \neg \text{Father}(x, z_1) \vee \neg \text{Father}(z_1, y), \\ L(x, y) \vee \neg \text{Ancestor}(x, z_2) \vee \neg \text{Father}(z_2, z_1) \vee \neg \text{Father}(z_1, y), \\ \dots$$

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

6 Conclusion

We have defined an inference rule and a strategy to generate the most general hypothesis allowing to infer a formula, represented by a single literal L, in the context of a given theory. This strategy is efficient in the sense that it generates only L-clauses. Then the only useless generated closes are those ones which are not minimal with regard to the subsumption. Moreover we have the feeling that this second step cannot take advantage of the work done in the first step, and

they can be executed into two independent steps without waste of efficiency.

Nevertheless many refinements of the strategy should be investigated in the future. One of them is to make use of an order on the predicate symbols.

An important issue we want to investigate in the future is the case of infinite answers. A first approach is to find a finite representation of those infinite sets, having some desirable properties. For example to be easy to understand. Another approach is to provide only partial answers, and to cut the computation in a “clean” state. The right approach certainly depends on the application field.

Our method can be considered as a step in order to supply new functionalities for automated deduction methods.

References

- [1] L. Cholvy. Answering queries addressed to a rule base. *Revue d'Intelligence Artificielle*, 4(1), 1990.
- [2] R.C.T. Lee C.L. Chang. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.
- [3] A. Kean and G. Tsiknis. An incremental method for generating prime implicants/implicates. *Journal of Symbolic Computation*, 9:185–206, 1990.
- [4] R. Kowalski. *Problems and Promises of Computational Logic*. Springer-Verlag, Brussels, 1990.
- [5] R.C.T. Lee. *A completeness theorem and a computer program for finding theorems derivable from given axioms*. PhD thesis, Univ. of California at Berkley, 1967.
- [6] M.Cayrol and P.Tayrac. Arc : un atms basé sur la résolution cat-correcte. *Revue d'Intelligence Artificielle*, 3(3), 1990.
- [7] Oxusoff and Rauzy. *Evaluation sémantique en Calcul Propositionnel*. PhD thesis, Université Aix-Marseille, 1989.
- [8] P.Siegel. *Représentation et utilisation de la connaissance en Calcul Propositionnel*. PhD thesis, Université de Aix-Marseille, 1987.
- [9] R. Reiter and de Kleer. Foundations of assumption-based truth maintenance system. In *AAAI-87*, 1987.