

AN APPROACH TO THE PROBLEM OF AVOIDING MISCONSTRUALS

Andrea S. Hemerly¹, Marco A. Casanova¹ and Antonio L. Furtado^{2,1}

¹Centro Científico Rio
IBM Brasil
Caixa Postal 4624
20.001, Rio de Janeiro, RJ - Brasil

²Departamento de Informática
Pontificia Universidade Católica do RJ
R. Marquês de S. Vicente, 225
22.453, Rio de Janeiro, RJ - Brasil

Abstract

When interacting with a database, a user is typically tempted to infer further information from that explicitly obtained from previous queries. However, his inferences are not necessarily valid, because his model of the world is often incomplete or even faulty. This paper addresses the problem of avoiding users' faulty inferences or misconstruals. It describes a cooperative interface to deductive databases that alters the processing of users' requests to include additional information that will block faulty inferences. The interface is governed by predefined user models, created during the design of the database. It also maintains a log of previous interactions with each user that records all information he has already acquired in his current session. The inferences of a given user are identified with those possible from the user model and the information in the log. The main result of the paper says that, given any user model and any deductive database, all logs produced by the interface indeed contain sufficient information to block all misconstruals.

1. INTRODUCTION

When interacting with a database, a user is typically tempted to infer further information from that explicitly obtained from previous queries. However, his inferences are not necessarily valid, because his model of the world is often incomplete or even faulty. For example, after consulting the database, a traveller may find out that his flight arrived on time and unadvisedly infer that it will depart on time, when in fact the flight has been delayed because the airport at the destination is closed. A more cooperative database system would have informed the traveller that the plane has indeed landed, assumed as the original question, and would have added that the flight has been delayed. To achieve cooperativeness, the system would naturally have some model of typical travellers.

We describe in this paper a cooperative interface to deductive databases that includes additional data along with the answers to the queries when it perceives that the user has gathered enough information that will lead to a faulty inference, sometimes also referred to as misconstrual [We].

The results described here are part of Project NICE [CF,HCF], whose purpose is to investigate cooperative query processing methods to reduce the cost of developing "help desks" and similar advanced database interfaces. Cooperative query processing has been explored, for example, in [BJ,CCL,CD], through the use of richer conceptual models, and in [Mo], via the generalization of failed queries. A natural language database query system, which recognizes users' presuppositions about the application domain, is also described in [Ka].

The problem of detecting and responding to plan-generation misconstruals is investigated in [Qu]. A good survey of user model techniques can be found in [KW].

This paper is organized as follows. Section 2 contains an informal introduction to the environment we propose for addressing the problem of misconstruals. Section 3 presents the logic programming concepts we will need in the paper. Section 4 formally introduces the environment and describes the cooperative interface. Section 5 proves the correctness of the cooperative interface. Section 6 summarizes the paper.

2. AN ENVIRONMENT FOR ADDRESSING THE PROBLEM OF MISCONSTRUALS

To address the problem of misconstruals, we propose a *cooperative interface* that passes additional information to the user when it discovers that he has gathered enough data to infer information that contradicts the database. The interface essentially simulates user's inferences and compares the result with what can be derived from the database.

The environment of the cooperative interface consists of a deductive database, a log and a user model. In addition, there are two theoretical concepts, called the certification criterion and the cooperativeness domain, created to guide the design and prove the correctness of the interface.

We assume that, in the context of a given session, the user remembers his past interactions with the deductive database and that he can use the information thus obtained in his (real world) inferences. The results of past interactions in the current session are kept in a *log*, which will indicate that certain facts hold and that certain other facts do not hold in the database.

For simplicity, we consider just one class of users, which isolates our problem from that of classifying users. Thus, from now on, when we refer to the user, we mean any user in this class. The *user model* is a theory, designed together with the database, that abstracts out the rules that the user adopts to reason about the domain of discourse in question. We stress that we use the term "user model" to mean a model of how the user reasons, which is somewhat different from the use of the term in the literature.

We model the user's inferences during a session by the deductions from the user model and the positive facts that the current log indicates to hold. In particular, we assume that the user reasons about negated facts only through *negation as finite failure*. This intuitively means that, in a given session, we model the inference of a negated fact $\neg A$, by the failure, in a finite number of steps, to find a proof for A from the user model and the facts that the log indicates to hold.

The *certification criterion* simply formalizes the contract between the user and the cooperative interface that governs the use of the log. It says that a deduction D is *certified* for a log λ iff, for any positive fact A used during the construction of D , the log λ does not indicate that A does not hold in the database. Note that the certification test is asymmetric in the sense that it depends only on the positive facts used. This follows because the cooperative interface can always block the derivation of a negative fact $\neg B$ by indicating, in the log, that B holds. Indeed, if this is the case, there will be a proof of B (since the log, together with the user model, indicates that B holds), which suffices to block the derivation of $\neg B$ by negation as finite failure.

The *cooperativeness domain* just defines a class of deductions. We agree that deductions outside the cooperativeness domain need not be simulated, which puts deliberate bounds on the level of cooperativeness required from the interface. In fact, we have to design the interface in such a way that, after it processes each query in a given session, for every deduction from the user model and the current information in the log, if the deduction is in the cooperativeness domain and it is certified for the current log, then its result is correct with respect to the database.

In this paper, we fix the cooperativeness domain as the set of all derivations of conjunctions of positive facts. Moreover, we assume that the user model is such that any positive fact derivable purely from the model (i.e., without looking at the log) is also derivable from the database. These choices are motivated by what we call the *log initialization problem*, which we now briefly discuss.

Let U be an (unrestricted) user theory, D be a deductive database and C be an (unrestricted) cooperativeness domain. Suppose that there is a fact A that follows from U alone by a deduction in C , but not from D . Then, we are in the presence of a misconstrual, which means that the cooperative interface has to inform the user that A does not hold as soon as he starts a session. This becomes even more cumbersome if we recall that the user may derive negative facts by negation as failure. In other words, if we do not restrict U or D , the interface will have to initialize the log, even before the user submits his first query, with a potentially enormous collection of positive facts. However, if C and U conform to the restrictions we imposed, then the initialization is unnecessary.

We conclude this section with two informal examples that indicate how the interface operates. The first example illustrates how to avoid misconstruals that arise when the user incorrectly invokes negation as finite failure for the lack of information.

Suppose that the user believes that a flight will always depart, if it arrived and it is not raining. This is equivalent to assuming a user model U that contains only one rule:

U.1. Flight x will depart, if x arrived and it is not raining

Consider that the deductive database has this same rule, that flight XY202 arrived and that the airport is closed due to bad weather. That is, let D be the following deductive database:

D.1. Flight x will depart, if x arrived and it is not raining

D.2. Flight XY202 arrived

D.3. It is raining

Suppose now that the user starts the dialog with the query:

Q. Has flight XY202 arrived?

The answer to Q therefore is YES. That is, at this point the user knows:

A₁. Flight XY202 arrived

If no extra information is passed to the user in the log, after the first query he will know fact A_1 , from which he can infer fact A_2 :

A₂. Flight XY202 will depart

However, the interface will discover that the user is about to wrongly infer fact A_2 as follows. It will first simulate a deduction R of A_2 from the user model U and A_1 . Then, the interface will analyse all steps of R and detect that R cannot be accepted because it is possible to infer the negation of fact A_3 :

A_3 . It is raining

from U and A_1 , by negation as finite failure, whereas it is not possible to infer the negation of A_3 from the database (since the database in fact includes A_3). Hence, the interface will include A_3 in the log to avoid the user's misconstrual. Indeed, the user can no longer infer A_2 using the complete information he obtained from the database (that is, A_1 and A_3).

The answer combined with the extra information in the log is roughly equivalent to the following English sentence:

Flight XY202 arrived and it is raining.

We stress that the misconstrual we just illustrated was caused by an incorrect use of negation as finite failure and that it could be blocked by including an additional fact in the log. Our next example illustrates a second type of misconstrual that arises when the user has inadequate rules.

Suppose that the user believes that a flight always departs, if it arrived. That is, let the user model now be V :

V.1. Flight x will depart, if x arrived

Assume the same deductive database D (including rule D.1 exactly as before). Then, the answer to Q remains unchanged, from which the user can again wrongly infer fact A_2 . The interface will again detect that A_2 does not follow from the database. The interface cannot block this misconstrual, however, by inserting additional facts in the log because the user's perception of the domain of discourse differs from that captured by the rules of the database. The interface will then act differently and include in the log an indication that A_2 is not deductible from the database.

The user can still infer A_2 from the answer to his query. However, his inference will not be certified for the current log, since the log indicates that A_2 does not hold.

The final answer will then be equivalent to the sentence:

Flight XY202 arrived but it will not depart.

We will formally define in section 4 the concepts here introduced intuitively. In particular, we will show that, in the context of logic programming, it is conceptually simple to simulate user's deductions and to detect misconstruals.

3. LOGIC PROGRAMMING BACKGROUND

In this section we recall some basic concepts of logic programming [L1].

An expression of the form $A \leftarrow B_1, \dots, B_n$ is a *program clause* or simply a *clause* iff A is a positive literal and B_1, \dots, B_n are literals. An expression of the form $\leftarrow B_1, \dots, B_n$ is a *normal goal* iff B_1, \dots, B_n are literals. The literals B_1, \dots, B_n are called the *body* of the program clause or of the normal goal and the literal A is called the *head* of the program clause. The empty clause \square is also considered to be a normal goal. Note that a normal goal $\leftarrow B_1, \dots, B_n$ represents the negation of $B_1 \wedge \dots \wedge B_n$.

A program clause is *allowed* iff every variable that occurs in the clause occurs in a positive

literal of its body. A normal goal G is *allowed* if every variable that occurs in G occurs in a positive literal of G .

A set P of program clauses intuitively contains only the if halves of the definitions of the predicate symbols. To deduce negative information from P , we have to add the only-if half of the definitions and a theory of equality. The resulting theory is called the *completion* of P and is denoted by $comp(P)$ [L1].

Let P be a set of program clauses and $\leftarrow Q$ be a normal goal. An *answer substitution* α for $\leftarrow Q$ from P is a substitution for the variables of $\leftarrow Q$. The answer α is *correct* iff $comp(P) \models \forall \bar{x}(Q\alpha)$, where \bar{x} is a list of the free variables of $Q\alpha$.

If G is a normal goal of the form $\leftarrow L_1, \dots, L_p$ and C is a program clause of the form $A \leftarrow M_1, \dots, M_q$, then we say that a normal goal G' is *derived* from G and C using a substitution θ iff θ is a most general unifier (mgu) of L_i and A and G' is the normal goal $\leftarrow (L_1, \dots, L_{i-1}, M_1, \dots, M_q, L_{i+1}, \dots, L_p)\theta$, where L_i is the literal selected from G by the selection function in question.

We now formally define a variation of SLDNF-resolution, called φ -SLDNF-resolution, that is the basis for both our cooperative interface and our model of how the user reasons and how he utilizes the information in the log.

Definition 1:

Let R be the set of all triples such that the first element is a finite sequence, with length $n+1$, of normal goals, the second element is a finite sequence, with length n , of substitutions and the third element is a finite sequence, also with length n , of clauses, for some $n > 0$. A *certification test* is a boolean function whose domain is R .

In the next sequence of definitions, let P be a set of program clauses and G be a non-empty normal goal. Without loss of generality, also let the selection function be that which selects the leftmost literal. We will then refer to the literal selected from a goal G as the first literal of G , and vice-versa. Finally, let φ be a certification test.

Definition 2:

A φ -SLDNF-derivation from $PU\{G\}$ is a triple S consisting of a sequence $G_0, G_1, \dots, G_i, \dots$ of normal goals, a sequence C_1, \dots, C_i, \dots of variants of clauses in P or negative literals and a sequence $\theta_1, \dots, \theta_i, \dots$ of mgu's such that $G_0 = G$ and, for each non-empty goal G_i of S , with $i \geq 0$, if G_i is of the form $\leftarrow L_1, \dots, L_m$ and L_1 is the literal selected from G_i , then:

- (i) if L_1 is a positive literal, then G_{i+1} is derived from G_i and C_{i+1} using θ_{i+1} ;
- (ii) if L_1 is a ground negative literal $\neg A$, then there is a finitely failed or uncertified φ -SLDNF-tree from $PU\{\neg A\}$. In this case, G_{i+1} is $\leftarrow L_2, \dots, L_m$, the substitution θ_{i+1} is the identity substitution, C_{i+1} is $\neg A$.

Definition 3:

Let S be a finite φ -SLDNF-derivation from $PU\{G\}$. Suppose that, for some $n \geq 0$, the sequence of goals of S is G_0, G_1, \dots, G_n and that the sequence of mgu's of S is $\theta_1, \dots, \theta_n$.

- (a) S has *rank* 0 iff only positive literals are selected; otherwise S has *rank* $k+1$ iff the largest rank of a φ -SLDNF-tree used to cancel a negative literal in S is k .

(b) S is *failed* iff G_n is non-empty and the literal selected from G_n is either a positive literal M and there is no clause in P whose head unifies with M , or a ground negative literal $\neg A$ and there is a certified φ -SLDNF-refutation from $PU\{\leftarrow A\}$.

(c) S is a φ -SLDNF-refutation from $PU\{G\}$ iff G_n is the empty goal.

The *answer to G from P computed by S* is the composition $\theta_1 \dots \theta_n$, restricted to the variables of G .

The φ -SLDNF-refutation S is *certified* (for φ) iff $\varphi(S) = \text{TRUE}$. Otherwise, S is *uncertified* (for φ).

Definition 4:

A φ -SLDNF-tree from $PU\{G\}$ is a tree T satisfying the following conditions:

- (i) the root is G ;
- (ii) each node of the tree is a normal goal;
- (iii) each branch forms a φ -SLDNF-derivation from $PU\{G\}$.

Assume now that T is finite. The tree T has *rank* k iff the largest rank of the φ -SLDNF-derivation corresponding to a branch of T is k . Moreover, T is *finitely failed or uncertified* iff each branch is a failed φ -SLDNF-derivation or an uncertified φ -SLDNF-refutation.

When φ is the trivial test that always returns TRUE, all refutations are certified. In this case, the above concepts for φ -SLDNF-resolution coincide with the standard concepts for SLDNF-resolution and we drop the reference to φ . In special, note that the definition of finitely failed or uncertified φ -SLDNF-trees differs from that of finitely failed SLDNF-trees to the extent that it allows branches that correspond to refutations, as long as they are uncertified. The reader should be aware that this departure from SLDNF-resolution implies that the process of checking for φ interferes with negation as failure.

We conclude with a simple proposition that states a property of allowed clauses we will use in the sequel (The proof follows directly from Proposition 15.1 of [LI]).

Proposition 1:

Let P be a set of allowed program clauses and G be an allowed normal goal. Let R be a φ -SLDNF-refutation from $PU\{G\}$ and assume that G_0, \dots, G_n is the sequence of goals and $\theta_1, \dots, \theta_n$ is the sequence of substitutions of R , for some $n \geq 0$.

- (a) the answer for G computed by R is ground.
- (b) for each $i \in [0, n)$, if A is the literal selected from G_i , then $A\theta_{i+1} \dots \theta_n$ is ground.

4. THE FORMAL MODEL

4.1 Deductive Databases and User Models

This section formally defines the concepts of deductive database, log and user model, along with the certification criterion and the cooperativeness domain, while section 4.2 describes the cooperative interface we propose.

A *deductive database* is a finite set D of allowed program clauses and a *query* over D is a non-empty allowed normal goal G .

A *log* λ is a set of expressions of the form $success(A)$ or $fail(A)$, where A is a positive ground literal. Intuitively, the cooperative interface will include $success(A)$ (or $fail(A)$) to inform the user that A (or $\neg A$) is deductible from the database.

A *user model* for D is a set U of program clauses such that: (i) each clause is in the language of D ; (ii) each clause is allowed; (iii) for any positive fact A , if $comp(U)$ logically implies A then $comp(D)$ also logically implies A . The first restriction is just a technical convenience and may be easily lifted, the second one is important for the adequacy of some of our key definitions, as discussed later on, and the reason from the third one has already been pointed out in section 2.

Let U be a user model and λ be a log. The *user state* induced by U and λ is the set $U[\lambda]$ obtained by adding to U all positive literals A such that $success(A) \in \lambda$.

We then identify user inferences in the presence of a log λ with the SLDNF-derivations from $U[\lambda]$ that do not contradict the information in λ . This is formalized by introducing a certification test $\varphi[\lambda]$ saying that, whenever the user needs to derive a positive fact A in the course of his reasoning, he must check whether the log λ does not indicate that A must be rejected, that is, whether $fail(A)$ is not in λ .

Recall that R is the set of all triples such that the first element is a finite sequence, with length $n+1$, of normal goals, the second element is a finite sequence, with length n , of substitutions and the third element is a finite sequence, also with length n , of clauses, for some n .

Definition 5:

The *certification test* for a log λ is the function $\varphi[\lambda]:R \rightarrow \{TRUE, FALSE\}$ defined as follows:

Let $S \in R$. Assume that G_0, \dots, G_n is the sequence of goals and $\theta_1, \dots, \theta_n$ is the sequence of substitutions of S , for some $n \geq 0$. Then, $\varphi[\lambda](S) = FALSE$ iff there is $i \in [0, n)$ such that the literal selected from G_i (the first literal by assumption) is a positive literal A and $fail(A\theta_{i+1} \dots \theta_n) \in \lambda$.

In this case, we also say that $A\theta_{i+1} \dots \theta_n$ is a $\varphi[\lambda]$ -*failure literal* of S and that G_i is a $\varphi[\lambda]$ -*failure goal* of S .

Furthermore, we say that S is *uncertified* for λ iff $\varphi[\lambda](S) = FALSE$.

The certification test for a log λ then formalizes the notion of certification criterion mentioned in section 2. Note that there is a natural ordering for the $\varphi[\lambda]$ -failure literals and the $\varphi[\lambda]$ -failure goals of S induced by the position of the goals in the sequence. In particular, we may refer to the last $\varphi[\lambda]$ -failure literal and the last $\varphi[\lambda]$ -failure goal of S .

Let U be a user model, λ be a log, and Q be a query. Then, by the restrictions on user models, logs and queries, $U[\lambda]$ is a set of allowed program clauses and Q is an allowed normal goal. Suppose that S is a $\varphi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{Q\}$. Hence, by Proposition 1(b), if the literal selected from a goal G_i of S is a positive literal A , then $A\theta_{i+1} \dots \theta_n$ is ground. Therefore, the test $fail(A\theta_{i+1} \dots \theta_n) \in \lambda$ is well-defined, since λ contains only ground expressions.

Given a log λ and a user model U , the user inferences are then identified with the set of all $\varphi[\lambda]$ -SLDNF-refutations from $U[\lambda]$ and a non-empty normal goal. As already discussed in section 2, we further identify the *cooperativeness domain* with the set of all $\varphi[\lambda]$ -SLDNF-refutations from $U[\lambda] \cup \{Q\}$ such that U is a user model, λ is a log and Q is a non-empty normal goal that contains only positive literals.

4.2 Description of a Cooperative Interface

In this section, we give a very high-level description of the cooperative interface we propose. The interface depends on a second variation of SLDNF-resolution, based on a different certification test, that checks, against the database, refutations from the user state.

Definition 6:

The *certification test for a deductive database* D is the function $\varphi[D]:R \rightarrow \{\text{TRUE}, \text{FALSE}\}$ defined as follows:

Let $S \in R$. Assume that G_0, \dots, G_n is the sequence of goals and $\theta_1, \dots, \theta_n$ is the sequence of substitutions of S , for some $n \geq 0$. Then, $\varphi[\lambda](S) = \text{FALSE}$ iff there is $i \in [0, n)$ such that

- either the literal selected from G_i is a positive literal A and there is a (standard) finitely failed SLDNF-tree from $DU\{\leftarrow A\theta_{i+1} \dots \theta_n\}$;
- or the literal selected from G_i is a negative ground literal $\neg B$, and there is a (standard) SLDNF-refutation from $DU\{\leftarrow B\}$.

In this case, we also say that $A\theta_{i+1} \dots \theta_n$, in the first case, and $\neg B$, in the second case, is a $\varphi[\lambda]$ -*failure literal* of S and that G_i is a $\varphi[\lambda]$ -*failure goal* of S .

Furthermore, we say that S is *uncertified for* λ iff $\varphi[\lambda](S) = \text{FALSE}$.

As for $\varphi[\lambda]$, we may refer to the last $\varphi[D]$ -failure literal and the last $\varphi[D]$ -failure goal of S .

Let U be a user model, λ be a log, and Q be a query. Then, $U[\lambda]$ is a set of allowed program clauses and Q is an allowed normal goal. Suppose that S is a $\varphi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{Q\}$. Again, by Proposition 1(b), if the literal selected from a goal G_i of S is a positive literal A , then $A\theta_{i+1} \dots \theta_n$ is ground. Therefore, testing if there is a (standard) finitely failed SLDNF-tree from $DU\{\leftarrow A\theta_{i+1} \dots \theta_n\}$ is simplified since the initial goal, $\leftarrow A\theta_{i+1} \dots \theta_n$, is always ground.

If A_1, \dots, A_p are the positive literals and $\neg B_1, \dots, \neg B_q$ the negative literals of a normal goal G , define $\text{expand}(G) = \{\text{success}(A_1), \dots, \text{success}(A_p), \text{fail}(B_1), \dots, \text{fail}(B_q)\}$. We are now ready to give a very high-level description of the cooperative interface.

A high-level description of the cooperative interface is given in Figure 1 below. Note that step 1 of the algorithm is subjected to the usual limitations of logic programming systems, in particular, that which says that it is in general undecidable to determine if a query will fail or not. Furthermore, note that, if λ is the current log, the algorithm simulates only the uncertified $\varphi[D]$ -SLDNF-refutations from $U[\lambda]$ starting on a goal $\leftarrow A$ such that A is the head of a clause in U . We will prove in the next section that this suffices to correctly avoid misconstruals in the cooperativeness domain.

Algorithm 1: Cooperative Interface

input: a query $\leftarrow Q$

output: an answer α for $\leftarrow Q$ or *FAIL*

parameters: a deductive database D , a user model U and a log λ

- 1) Find an answer substitution α for $\leftarrow Q$ from D ;
if the query fails then return *FAIL* and stop.
 - 2) Add all expressions in $expand(Q\alpha)$ to λ .
 - 3) While λ changes do:
 - a) Simulate all uncertified $\varphi[D]$ -SLDNF-refutations from $U[\lambda] \cup \{\leftarrow A \mid A \leftarrow B_1, \dots, B_m \in U\}$.
 - b) For each such refutation R ,
 - i) if the last $\varphi[D]$ -failure literal of R is a positive literal A , then add *fail*(A) to λ ;
 - ii) if the last $\varphi[D]$ -failure literal of R is a negative literal $\neg B$, then add *success*(B) to λ ;
 - 4) Return α .
-

During a session, the user may then pose several queries to the system. Each query is then passed to the cooperative interface, along with the log produced by the processing of the previous query (the log at the beginning of a session is always empty). The behaviour of the system during a session then induces a *cooperative dialog* C , defined as a sequence of triples $(Q_1, \alpha_1, \lambda_1), \dots, (Q_n, \alpha_n, \lambda_n)$ where, for each $i \in [1, n]$, Q_i is a query, α_i is an answer substitution for Q_i or the expression *FAIL*, and λ_i is a log. Note that, for each $i \in [1, n]$, $\lambda_{i-1} \subseteq \lambda_i$ and $expand(Q_i \alpha_i) \subseteq \lambda_i$, if α_i is not *FAIL*.

5. THEORETICAL RESULTS

We prove in this section that the cooperative interface is correct in the sense that, for any log it creates, if F can be obtained from the user model and the log by a deduction in the cooperativeness domain, then F is true in the database. More precisely, recall that $comp(P)$ denotes the completion of a normal program P and that $U[\lambda]$ denotes the set obtained by adding to a user model U all positive literals A such that *success*(A) occurs in a log λ .

Definition 7:

Let U be a user model, D be a deductive database and E be a cooperativeness domain. A log λ is *sound for D and U with respect to E* iff, if R is a $\varphi[\lambda]$ -SLDNF-refutation in E from $U[\lambda]$ and a goal $\leftarrow G$, and if α is the answer computed by R for $\leftarrow G$, then $comp(D) \models \forall \bar{x}(G\alpha)$, where \bar{x} is a list of the free variables of $G\alpha$.

In the next sequence of results, let U be a user model, D be a deductive database, $\leftarrow G$ be a goal containing only positive literals, and D be the cooperativeness domain introduced in section 4, that is, the set of all refutations starting on a goal containing only positive literals. Also let C be a dialog produced by Algorithm 1 and λ be a log in the dialog. Note that any $\varphi[\lambda]$ -SLDNF-refutation R from $U[\lambda] \cup \{\leftarrow G\}$ is then in D , since $\leftarrow G$ contains only positive literals.

To prove the correctness of the interface, we need three auxiliary results. The first one is a direct consequence of the definitions in section 4:

Proposition 2:

- (a) If $\text{success}(A) \in \lambda$ then there is an SLDNF-refutation from $\text{DU}\{\leftarrow A\}$.
- (b) If $\text{fail}(A) \in \lambda$ then there is a finitely failed SLDNF-tree from $\text{DU}\{\leftarrow A\}$.

The second result establishes a tight relationship between user's derivations in the presence of a log and the derivations simulated by the interface.

Lemma 1:

- (a) There is a certified $\varphi[\lambda]$ -SLDNF-refutation from $\text{U}[\lambda]\text{U}\{\leftarrow G\}$ iff there is also a certified $\varphi[D]$ -SLDNF-refutation from $\text{U}[\lambda]\text{U}\{\leftarrow G\}$, and both compute the same answer.
- (b) There is a finitely failed or uncertified $\varphi[\lambda]$ -SLDNF-tree from $\text{U}[\lambda]\text{U}\{\leftarrow G\}$ iff there is also a finitely failed or uncertified $\varphi[D]$ -SLDNF-tree $\text{U}[\lambda]\text{U}\{\leftarrow G\}$.

The last auxiliary lemma shows that every answer computed by a certified $\varphi[D]$ -SLDNF-refutation is sound with respect to $\text{comp}(D)$.

Lemma 2:

If R is a certified $\varphi[D]$ -SLDNF-refutation from $\text{U}[\lambda]\text{U}\{\leftarrow G\}$, and if α is the answer computed by R for $\leftarrow G$, then $\text{comp}(D) = G\alpha$.

Note that, by Proposition 1(a), $G\alpha$ is indeed ground. We are now ready to state the major result of this section:

Theorem 1: (Correctness of the Cooperative Interface)

Let C be a dialog produced by Algorithm 1. Then, any log λ in C is sound for D and U with respect to the cooperativeness domain D .

Therefore, we may assert that Algorithm 1 correctly avoids all misconstruals that derive conjunctions of positive literals, even though the interface simulates only refutations that start on a goal $\leftarrow A$ such that A is the head of a clause in the user model.

We conclude this section with a second result of interest showing that, given any two logs, λ and γ , of a dialog C produced by Algorithm 1, anything that can be deduced from $\text{U}[\lambda]$ can also be deduced from $\text{U}[\gamma]$, if γ is generated after λ . That is, the sequence of logs produced by the cooperative interface induces monotonic behavior.

Theorem 2:

Let C be a dialog produced by Algorithm 1 and let λ and γ be two logs in C such that γ is generated after λ . Let $\leftarrow G$ be a goal containing only positive literals. If there is a certified $\varphi[\lambda]$ -SLDNF-refutation from $\text{U}[\lambda]\text{U}\{\leftarrow G\}$, then there is also a certified $\varphi[\gamma]$ -SLDNF-refutation from $\text{U}[\gamma]\text{U}\{\leftarrow G\}$.

6. CONCLUSIONS

We first described a model for users' inferences capturing the intuition that, whenever the user needs to derive a positive fact A in his inference, he must check whether the current log does not indicate that A must be rejected. Then, we presented a cooperative interface that is responsible for all the inferences from the deductive database that are necessary to answer the users' queries and to compute what additional information to include in the log to avoid misconstruals. Therefore, we may view the interface as summarizing for the user the information in the deductive database and passing it in the log. Finally, we proved that the cooperative interface is correct in the sense that any log produced during a dialog contains enough information to avoid misconstruals.

We are now experimenting with cooperativeness domains that limit the derivations that the interface must simulate to those satisfying a certain complexity constraint (a simple example would be to limit the length of the derivation). We will also test the idea of limiting the domain by capturing in advance the goal(s) that the user wants to achieve [AP]. In the context of Project NICE, we also explored a simpler approach, based on request modification rules compiled from explicit patterns of user inferences [HCF].

REFERENCES

- [AP] J. F. Allen and C. R. Perrault, "Analyzing intentions in utterances", *Artificial Intelligence* 15:3 (1980), 143-178.
- [BJ] L. Bolc and M. Jarke (eds.), *Cooperative Interfaces to Information Systems*, Springer-Verlag (1986).
- [CCL] W. Chu, Q. Chen and R-C. Lee, "Cooperative Query Answering via Type Abstraction Hierarchy", Proc. Int. Working Conference on Cooperating Knowledge based Systems, Univ. Keele, UK (1990).
- [CF] M. A. Casanova and A. L. Furtado, "An Information System Environment based on Plan Generation", Proc. Int. Working Conference on Cooperating Knowledge based Systems, Keele, UK (1990).
- [CD] F. Cuppens and R. Demolombe, "Cooperative answering: a methodology to provide intelligent access to databases", Proc. of the Second International Conference on Expert Database Systems, L. Kerschberg (ed.), Benjamin/Cummings (1989), 621-643.
- [HCF] A. S. Hemerly, M. A. Casanova and A. L. Furtado, "Cooperative behaviour through request modification", (accepted to the 10th Int'l. Conf. on the Entity-Relationship Approach, San Mateo, CA, USA (Oct. 23-25, 1991)).
- [Ka] S. J. Kaplan, "Cooperative Responses from a Portable Natural Language Query System", *Artificial Intelligence* 19:2 (1982), 165-187.
- [KW] A. Kobsa and W. Wahlster (eds.), *User Models in Dialog Systems*, Springer-Verlag (1989).
- [LI] J.W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag (1987).
- [Mo] A. Motro, "Query generalization: a technique for handling query failure", Proc. First International Workshop on Expert Database Systems (1984), 314-325.
- [Qu] A. Quilici, "Detecting and Responding to Plan-Oriented Misconceptions", in *User Models in Dialog Systems*, A. Kobsa and W. Wahlster (eds.), Springer-Verlag (1989).
- [We] B.L. Webber, "Questions, answers and responses: interacting with knowledge base systems", in *On knowledge base management systems*, M.L. Brodie and J. Mylopoulos (eds.) - Springer (1986).