

Exploiting user models to avoid misconstruals

Andrea S. Hemerly, Marco A. Casanova, and Antonio L. Furtado

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 chapter 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 chapter 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.

In this chapter we describe 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 [Web86].

The results described here are part of Project NICE [CF90, HCF91a], 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 [BJ86, CCL90, CD89], through the use of richer conceptual models, and in [Mot84] via the generalization of failed queries. A natural language database query system, which recognizes users' presuppositions about the application domain, is also described in [Kap82]. The problem of detecting and responding to plan-generation misconstruals is investigated in [Qui89]. A good survey of user model techniques can be found in [KW89].

This work 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 that we will need in the chapter. Section 4 formally introduces the environment and describes the cooperative interface. Section 5 proves the correctness of the cooperative interface. Section 6 summarizes the chapter.

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 the user's inferences and compares the result with what can be inferred from the database.

In this work, the environment of the cooperative interface consists of a deductive database, a log, and a user model. The generalization of this environment, including a more sophisticated user/domain model [CF90, HCF91a], is simple. 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. In this chapter, we adopt a certification criterion that accepts a deduction D as *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 criterion 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 chapter, 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 discuss briefly.

Let U be an (unrestricted) user theory, D be a deductive database, and \mathcal{E} be an (unrestricted) cooperativeness domain. Suppose that there is a fact A that follows from U alone by a deduction in \mathcal{E} 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 \mathcal{E} 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 \mathcal{E} 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 dialogue with the query:

Q. Has flight XY202 arrived?

The answer to Q is therefore 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 infer fact A_2 wrongly 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 requires inferring fact A_3 :

A₃. It is not raining

from U and A_1 , by negation as finite failure, whereas it is not possible to infer A_3 from the database (since the database in fact includes $D.3$). Hence, the interface will include $D.3$ 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 $D.3$).

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.

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

3 Logic programming background

In this section we recall some basic concepts of logic programming [Llo87].

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 .

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 $\leftarrow Q$.

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 3.1 Let \mathcal{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 \mathcal{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 3.2 A Φ -SLDNF-derivation from $P \cup \{G\}$ 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 $P \cup \{\leftarrow A\}$. 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 3.3 Let S be a finite Φ -SLDNF-derivation from $P \cup \{G\}$. 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 $P \cup \{\leftarrow A\}$.
- (c) S is a Φ -SLDNF-refutation from $P \cup \{G\}$ 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 $\Phi(S) = \text{TRUE}$. Similarly, the Φ -SLDNF-refutation S is *uncertified* (for Φ) iff $\Phi(S) = \text{FALSE}$.

Definition 3.4 A Φ -SLDNF-tree from $P \cup \{G\}$ 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 $P \cup \{G\}$.

Assume that T is finite. Then, 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 particular, 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, as the following simple example illustrates.

Example 3.5 Let Φ be the certification test such that $\Phi(S)=\text{TRUE}$ iff all substitutions in S replace each variable by a ground term. Let $P = \{p(x) \leftarrow q(x, y), q(a, z)\}$. Hence, there is a Φ -SLDNF-refutation R from $P \cup \{\leftarrow \neg p(a)\}$ because there is a finitely failed or uncertified Φ -SLDNF-tree T from $P \cup \{\leftarrow p(a)\}$:

R :	1. $\leftarrow \neg p(a)$ 2. \square success	T :	$\leftarrow p(a)$ $\quad $ $\leftarrow q(a, y)$ $\quad $ $\quad \square$ uncertified (by assumption on Φ)
-------	--	-------	---

But there is no (standard) SLDNF-refutation from $P \cup \{\leftarrow \neg p(a)\}$, since the SLDNF-tree T' from $P \cup \{\leftarrow p(a)\}$ has a success branch. Indeed, R' is the only SLDNF-derivation from $P \cup \{\leftarrow \neg p(a)\}$, but R' is failed:

R' :	1. $\leftarrow \neg p(a)$ fail	T' :	$\leftarrow p(a)$ $\quad $ $\leftarrow q(a, y)$ $\quad $ $\quad \square$ success
--------	-----------------------------------	--------	---

We conclude with a simple proposition that states a property of allowed clauses that we will use in the sequel.

Proposition 3.6 *Let P be a set of allowed program clauses and G be an allowed normal goal. If R is a Φ -SLDNF-refutation from $P \cup \{G\}$ then the answer R computes is ground.*

(The proof follows directly from Proposition 15.1 of [Llo87]).

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 there is an SLDNF-refutation from $U \cup \{\leftarrow A\}$ then there is no finitely failed SLDNF-tree from $D \cup \{\leftarrow A\}$. 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 is the log initialization problem, already discussed 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 $\Phi[\lambda]$ 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 \mathcal{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 4.1 The *certification test for a log λ* is the function $\Phi[\lambda] : \mathcal{R} \mapsto \{\text{TRUE}, \text{FALSE}\}$ defined as follows:

Let $S \in \mathcal{R}$. 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, $\Phi[\lambda](S) = \text{FALSE}$ 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 $\Phi[\lambda]$ -*failure literal* of S and that G_i is a $\Phi[\lambda]$ -*failure goal* of S .

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

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 $\Phi[\lambda]$ -failure literals and the $\Phi[\lambda]$ -failure goals of S induced by the position of the goals in the sequence. In particular, we may refer to the last $\Phi[\lambda]$ -failure literal and the last $\Phi[\lambda]$ -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 $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{Q\}$. Hence, by Proposition 3.6, 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 $\Phi[\lambda]$ -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 $\Phi[\lambda]$ -SLDNF-derivations 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.

The following example formalizes one of the examples given in Section 2 and illustrates the concepts introduced in this section:

Example 4.2 Let U be a user model that contains only one clause:

U.1. $depart(x, f(t)) \leftarrow arrived(x, t)$

Let λ be the following log:

$$\lambda = \{success(arrived(XY202, 10:00am), \\ fail(depart(XY202, f(10:00am))))\}$$

Then, the current user state $U[\lambda]$ contains the following clauses:

U.1. $depart(x, f(t)) \leftarrow arrived(x, t)$
 U.2. $arrived(XY202, 10:00am)$

Consider the following sequence R of goals:

R.1. $\leftarrow depart(XY202, f(10:00am))$
 R.2. $\leftarrow arrived(XY202, 10:00am)$
 R.3. \square

Then, R is a (standard) SLDNF-refutation from $U[\lambda] \cup \{\leftarrow depart(XY202, f(10:00am))\}$ (that is, a certified Φ -SLDNF-refutation, where Φ is the test that always returns true). However, R is an uncertified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{\leftarrow depart(XY202, f(10:00am))\}$ because $depart(XY202, f(10:00am))$ was selected in $R.1$, but $fail(depart(XY202, f(10:00am)))$ is in λ .

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 refutations from the user state against the database.

Definition 4.3 The *certification test for a deductive database D* is the function $\Phi[D] : \mathcal{R} \mapsto \{\text{TRUE}, \text{FALSE}\}$ defined as follows:

Let $S \in \mathcal{R}$. 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, $\Phi[D](S) = \text{FALSE}$ 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 finitely failed SLDNF-tree from $D \cup \{\leftarrow A\theta_{i+1} \dots \theta_n\}$;
- or the literal selected from G_i is a negative ground literal $\neg B$, and there is an SLDNF-refutation from $D \cup \{\leftarrow B\}$.

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 $\Phi[D]$ -failure literal of S and that G_i is a $\Phi[D]$ -failure goal of S .

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

As for $\Phi[\lambda]$, we may refer to the last $\Phi[D]$ -failure literal and the last $\Phi[D]$ -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 $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{Q\}$. Again, by Proposition 3.6, 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 $D \cup \{\leftarrow A\theta_{i+1} \dots \theta_n\}$ 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

$$\text{expand}(G) = \{\text{success}(A_1), \dots, \text{success}(A_p), \text{fail}(B_1), \dots, \text{fail}(B_q)\}.$$

A high-level description of the cooperative interface is given in Figure 1. Note that steps 1 and 3a of the procedure are 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 (see also Example 4.6 for further discussions). Moreover, note that, if λ is the current log, the procedure simulates only the $\Phi[D]$ -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.

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 $\Phi[D]$ -SLDNF-refutations
 from $U[\lambda] \cup \{\leftarrow A \mid A \leftarrow B_1, \dots, B_m \in U\}$.
 - (b) If R is uncertified then:
 - i. if the last $\Phi[D]$ -failure literal of R is a positive literal A ,
 then add $fail(A)$ to λ ;
 - ii. if the last $\Phi[D]$ -failure literal of R is a negative literal $\neg B$,
 then add $success(B)$ to λ ;
 4. Return α .
-

Figure 1: The cooperative interface - procedure 1

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*.

We now give two examples that indicate how the interface operates and that formalize the examples given in Section 2.

Example 4.4 Let U be a user model that contains only one clause:

$$U.1. \quad \text{depart}(x, f(t)) \leftarrow \text{arrived}(x, t), \neg \text{rain}(f(t))$$

Let D be the following deductive database:

$$D.1. \quad \text{depart}(x, f(t)) \leftarrow \text{arrived}(x, t), \neg \text{rain}(f(t))$$

$$D.2. \quad \text{arrived}(XY202, 10:00am)$$

$$D.3. \quad \text{rain}(f(10:00am))$$

Suppose that the user starts a new session with the query:

$$Q. \quad \leftarrow \text{arrived}(XY202, t)$$

Recall that the log λ at the beginning of a section is empty by assumption. The interface will compute the following answer substitution for Q :

$$\alpha = \{t/10:00am\}$$

and add $\text{expand}(Q\alpha)$ to the current log, which then becomes:

$$\lambda = \{\text{success}(\text{arrived}(XY202, 10:00am))\}$$

The user state at this point is the theory:

$$U[\lambda] = U \cup \{\text{arrived}(XY202, 10:00am)\}$$

from which it is possible to infer $\text{depart}(XY202, f(10:00am))$. The interface proceeds to simulate refutations. In particular, it will simulate the following uncertified $\Phi[D]$ -refutation R from $U[\lambda] \cup \{\leftarrow \text{depart}(x, f(t))\}$:

$$R.1. \quad \leftarrow \text{depart}(x, f(t))$$

$$R.2. \quad \leftarrow \text{arrived}(x, f(t)), \neg \text{rain}(f(t))$$

$$R.3. \quad \leftarrow \neg \text{rain}(f(10:00am))$$

$$R.4. \quad \square$$

whose $\Phi[D]$ -failure literal is $\neg \text{rain}(f(10:00am))$ because there is a trivial (standard) SLDNF-refutation from $D \cup \{\leftarrow \text{rain}(f(10:00am))\}$. Hence, the interface will include the expression $\text{success}(\text{rain}(f(10:00am)))$ in the log, which becomes:

$$\lambda = \{\text{success}(\text{arrived}(XY202, 10:00am)), \text{success}(\text{rain}(f(10:00am)))\}$$

The interface then stops and returns the answer α and, implicitly, the final value of the log λ . The final user state is the theory:

$$U[\lambda] = U \cup \{\text{arrived}(XY202, 10:00am), \text{rain}(f(10:00am))\}$$

from which it is no longer possible to infer $\text{depart}(XY202, f(10:00am))$.

Example 4.5 Suppose that the user model now is V :

$$V.1. \quad \text{depart}(x, f(t)) \leftarrow \text{arrived}(x, t)$$

Assume the same deductive database D and the same query Q as before, which means that the answer substitution α for Q remains unchanged. The log and the user state immediately before the interface begins simulating refutations is:

$$\lambda = \{\text{success}(\text{arrived}(XY202, 10:00am))\}$$

$$\bar{V}[\lambda] = V \cup \{\text{arrived}(XY202, 10:00am)\}$$

The interface will simulate, among others, the following uncertified $\Phi[D]$ -refutation S from $\bar{V}[\lambda] \cup \{\leftarrow \text{depart}(x, f(t))\}$:

$$\begin{array}{ll} S.1. & \leftarrow \text{depart}(x, f(t)) \\ S.2. & \leftarrow \text{arrived}(x, t) \qquad \theta_1 = \varepsilon \\ S.3. & \square \qquad \theta_2 = \{x/XY202, t/10:00am\} \end{array}$$

whose $\Phi[D]$ -failure literal is $(\text{depart}(x, f(t)))\theta_1\theta_2$, that is, $\text{depart}(XY202, f(10:00am))$, because there is a (standard) finitely failed SLDNF-tree from $D \cup \{\leftarrow \text{depart}(XY202, f(10:00am))\}$. Hence, the interface will include the expression $\text{fail}(\text{depart}(XY202, f(10:00am)))$ in the log. The log returned after the query and the final user state will be:

$$\lambda = \{success(arrived(XY202, 10:00am)), \\ fail(depart(XY202, f(10:00am)))\}$$

$$\bar{V}[\lambda] = V \cup \{arrived(XY202, 10:00am)\}$$

Note that there is a $\Phi[\lambda]$ -SLDNF-refutation from $\bar{V}[\lambda] \cup \{\leftarrow depart(XY202, f(10:00am))\}$, but it will not be certified for $\Phi[\lambda]$, since λ contains $fail(depart(XY202, f(10:00am)))$. This intuitively means that the extra information in the log suffices to warn the user that he must reject this (faulty) inference.

We close this section with a third example that illustrates how to compute the failure literal of a refutation easily. The key point is to instrument the clauses of the user model with *belief literals*, which are literals enclosed in square brackets that help trace the literals cancelled in refutations. The standard selection function is also altered to select the leftmost (normal) literal, if there is one, or the leftmost belief literal, if there are only belief literals in the goal.

Note that the implementation of the $\Phi[D]$ certification test is subjected to the usual limitations of logic programming systems.

Example 4.6 Let U , D , and Q be as in Example 4.4. We instrument the only clause in U as follows:

$$U.1 \quad depart(x, f(t)) \leftarrow arrived(x, t), \neg rain(f(t)), [\neg rain(f(t))], \\ [depart(x, f(t))]$$

The refutation R now becomes the sequence:

- R.1. $\leftarrow depart(x, f(t))$
- R.2. $\leftarrow arrived(x, f(t)), \neg rain(f(t)), [\neg rain(f(t))], [depart(x, f(t))]$
- R.3. $\leftarrow \neg rain(f(10:00am)), [\neg rain(f(10:00am))], [depart(XY202, \\ f(10:00am))]$
- R.4. $\leftarrow [\neg rain(f(10:00am))], [depart(XY202, f(10:00am))]$

The selection of the negative belief literal $[\neg rain(f(10:00am))]$ from $R.4$ signals that the interface must try to find a (standard) SLDNF-refutation from $D \cup \{\leftarrow rain(f(10:00am))\}$.

Since the search is successful, the $\Phi[D]$ -failure literal of the original refutation R is $\neg rain(f(10:00am))$.

Now, let V be the user model introduced in Example 4.5. We instrument V with belief literals as follows:

V.1. $depart(x, f(t)) \leftarrow arrived(x, t), [depart(x, f(t))]$

which means that the refutation S becomes:

S.1. $\leftarrow depart(x, f(t))$
 S.2. $\leftarrow arrived(x, f(t)), [depart(x, f(t))]$
 S.3. $\leftarrow [depart(XY202, f(10:00am))]$

This time, the selection of the positive belief literal $[depart(XY202, f(10:00am))]$ from $S.3$ signals that the interface must try to find a (standard) finitely failed SLDNF-tree from $D \cup \{\leftarrow depart(XY202, f(10:00am))\}$. Since the search is successful, the $\Phi[D]$ -failure literal of S is $depart(XY202, f(10:00am))$.

5 Theoretical results

5.1 Soundness of the cooperative interface

In this section we prove that the cooperative interface is correct in the sense that, for any log it creates, if a fact F can be obtained from the user model and the log by a deduction in the cooperativeness domain, then we cannot detect using finite failure that F is false in the database. Recall 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 5.1 Let U be a user model, D be a deductive database, and \mathcal{E} be a cooperativeness domain. A log λ is *finitely sound for D and U with respect to \mathcal{E}* iff, if R is a certified $\Phi[\lambda]$ -SLDNF-refutation R in \mathcal{E} from $U[\lambda]$ and a goal $\leftarrow G$, and if α is the answer computed by R for $\leftarrow G$, then there is no finitely failed SLDNF-tree from $D \cup \{\leftarrow G\alpha\}$.

In the next sequence of results, let U be a user model, D be a deductive database, $\leftarrow G$ be a query containing only positive literals, and \mathcal{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 dialogue produced by Procedure 1, and λ be a log in the dialogue. Note that any $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{\leftarrow G\}$ is then in \mathcal{D} , since $\leftarrow G$ contains only positive literals.

To prove the correctness of the interface, we need some auxiliary results. The first one is a direct consequence of the definitions in Section 4 and Procedure 1:

Proposition 5.2 *Let A be a ground positive literal. Then:*

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

The next corollary follows directly from the above proposition and the definition of SLDNF-resolution.

Corollary 5.3 *Let A be a ground positive literal. Thus, if $\text{success}(A) \in \lambda$ then $\text{fail}(A) \notin \lambda$.*

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

Lemma 5.4

- (a) If S is a $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$ in \mathcal{D} , then S is also a $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$.
- (b) If S is a failed $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$ in \mathcal{D} , then S is also a failed $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$.
- (c) If S is a certified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$ in \mathcal{D} , then S is also a certified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$.
- (d) If S is an uncertified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$ in \mathcal{D} , then S is also an uncertified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$.
- (e) If N is a finitely failed or uncertified $\Phi[\lambda]$ -SLDNF-tree from $U[\lambda] \cup \{G\}$ in \mathcal{D} , then N is also a finitely failed or uncertified $\Phi[D]$ -SLDNF-tree $U[\lambda] \cup \{G\}$.

PROOF. The proof will be by induction on the rank of SLDNF-derivations, certified or uncertified SLDNF-refutations and finitely failed or uncertified SLDNF-trees. Recall that, since G contains only positive literals, any $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$ is in \mathcal{D} .

Basis: Assume that the rank is 0.

(a) Let S be a $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$ of rank 0, that is, no negative literal is selected. Since only positive literals are selected in S , the $\Phi[\lambda]$ and $\Phi[D]$ certification tests have no influence on S . Hence, a $\Phi[\lambda]$ -derivation of rank 0 is also a $\Phi[D]$ -derivation of rank 0.

(b) Let S be a failed $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$ of rank 0. As in (a), S is trivially a failed $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$ of rank 0.

(c) Let S be a certified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$ of rank 0. By (a), S is also a $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$. To prove

that S is a certified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$, we will show that the $\Phi[D]$ -certification test succeeds, that is, there are no $\Phi[D]$ -failure goals for S .

Assume that S has length n , the sequence of goals is $G_0 = G, \dots, G_n$, the sequence of variants of clauses in $U[\lambda]$ is C_1, \dots, C_n , and the sequence of mgu's is $\theta_1, \dots, \theta_n$. Let $i \in [0, n)$. Assume that L_1 is the literal cancelled from G_i . Since S has rank 0, L_1 is a positive literal. Since the $\Phi[\lambda]$ -certification test succeeds, we have $fail(L) \notin \lambda$, for $L = L_1\theta_{i+1} \dots \theta_n$. Assume that A is the head of C_{i+1} . Thus, we may construct from S a $\Phi[D]$ -SLDNF-refutation S' from $U[\lambda] \cup \{\leftarrow A\}$, such that S' is simulated in the same iteration that Procedure 1 generates λ , the composition of mgu's is α and $A\alpha = L$. Thus, since $fail(L) \notin \lambda$, by Definition 4.3 and step 3 of Procedure 1, there is no finitely failed SLDNF-tree from $D \cup \{\leftarrow L\}$. Hence, G_i is not a $\Phi[D]$ -failure goal.

Therefore, there are no $\Phi[D]$ -failure goals for S .

(d) Let S be an uncertified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$ of rank 0. By (a), S is also a $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$. To prove that S is an uncertified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$, we will show that the $\Phi[D]$ -certification test fails, that is, there is a $\Phi[D]$ -failure goal for S .

Assume that S has length n , the sequence of goals is $G_0 = G, \dots, G_n$ and the sequence of mgu's is $\theta_1, \dots, \theta_n$. Suppose that G_i is the $\Phi[\lambda]$ -failure goal in S , for some $i \in [0, n)$. Since S has rank 0, the literal selected from G_i is a positive literal, say M , and $fail(M\theta_{i+1} \dots \theta_n) \in \lambda$. By Proposition 5.2, there is a finitely failed SLDNF-tree from $D \cup \{\leftarrow M\theta_{i+1} \dots \theta_n\}$. Hence, G_i is also a $\Phi[D]$ -failure goal in S .

(e) Let N be a finitely failed or uncertified $\Phi[\lambda]$ -SLDNF-tree N of rank 0 from $U[\lambda] \cup \{G\}$. By Definitions 3.4 and 4.1, each branch in N is a failed $\Phi[\lambda]$ -SLDNF-derivation or an uncertified $\Phi[\lambda]$ -SLDNF-refutation, of rank 0. By (b) and (d), N is trivially a finitely failed or uncertified $\Phi[D]$ -SLDNF-tree.

Induction step: Let $k > 0$. Suppose that the results hold for all refutations and trees of rank $j < k$.

(a) Let S be a $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$ of rank k . Assume that S has length n , and that the sequence of goals is $G_0 = G, \dots, G_n$. We show that S is also a $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$.

Let $j \in [1, n]$. Consider that L_1 is the literal cancelled in G_{j-1} . If L_1 is a positive literal the proof is the same as for the basis step. Suppose then that L_1 is a ground negative literal $\neg L$. Since $\neg L$ is cancelled in S , it means that there is a finitely failed or uncertified $\Phi[\lambda]$ -SLDNF-tree from $U[\lambda] \cup \{\leftarrow L\}$ of rank less than k . By the induction hypothesis, there is also a finitely failed or uncertified $\Phi[D]$ -SLDNF-tree from $U[\lambda] \cup \{\leftarrow L\}$.

Then, $\neg L$ can indeed be cancelled in G_{j-1} deriving G_j .

(b) Let S be a finitely failed $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$ of rank k . By (a), S is also a $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$. To prove that S is a failed $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$, we show that the literal selected in the last goal of S cannot be cancelled.

Assume that S has length n and the sequence of goals is $G_0 = G, \dots, G_n$. Suppose that G_n is a goal of the form $\leftarrow L_1, \dots, L_m$. If L_1 is a positive literal, the proof is the same as for (b) in the basis step. Suppose then that L_1 is a ground negative literal $\neg L$. Since $\neg L$ is not cancelled in G_n , there is a certified $\Phi[\lambda]$ -SLDNF-refutation of rank less than k from $U[\lambda] \cup \{\leftarrow L\}$. Then, by the induction hypothesis, there is also a certified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{\leftarrow L\}$. Thus, L_1 cannot be cancelled in G_n , that is, S is indeed a failed $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$.

(c) Let S be a certified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$ of rank k . By (a), S is also a $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$. To prove that S is a certified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$, we will show that the $\Phi[D]$ -certification test succeeds, that is, there are no $\Phi[D]$ -failure goals for S .

Assume that S has length n , the sequence of goals is $G_0 = G, \dots, G_n$ and the sequence of variants of clauses in $U[\lambda]$ is C_1, \dots, C_n . Let $i \in [0, n]$. Assume that L_1 is the literal cancelled from G_i . If L_1 is a positive literal, the proof is the same as for (c) in the basis step. Suppose then that L_1 is a ground negative literal $\neg L$. Since $\neg L$ is cancelled in S , we have that $\text{success}(L) \notin \lambda$. Indeed, if $\text{success}(L) \in \lambda$ then, by definition of user state, $L \in U[\lambda]$ and, by Corollary 5.3, $\text{fail}(L) \notin L$. Thus, there would be a certified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{\leftarrow L\}$, which would imply that $\neg L$ could not be cancelled in S . Hence, we conclude that $\text{success}(L) \notin \lambda$. Also, since G contains only positive literals, the negative literal $\neg L$ is introduced in S by a variant of a clause in U , say C_q , for some $q \in [1, i]$ with $i > 0$. Assume that A is the head of C_q . We may construct from S a $\Phi[D]$ -SLDNF-refutation S' from $U[\lambda] \cup \{\leftarrow A\}$, which is simulated by Procedure 1; in the same iteration it generates λ , and $\neg L$ is cancelled in S' . Thus, since $\text{success}(L) \notin \lambda$, by Definition 4.3 and step 3 of Procedure 1, there is no SLDNF-refutation from $D \cup \{\leftarrow L\}$. Hence, G_i is not a $\Phi[D]$ -failure goal. Therefore, there are no $\Phi[D]$ -failure goals for S .

(d) Let S be an uncertified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$ of rank k . By (a), S is also a $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$. To prove that S is an uncertified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$, we will show now that the $\Phi[D]$ -certification test fails for S , and the $\Phi[D]$ -failure literal is a positive literal.

Assume that S has length n , the sequence of goals is $G_0 = G, \dots, G_n$ and the sequence of variants of clauses in $U[\lambda]$ is C_1, \dots, C_n . Let G_i be the $\Phi[\lambda]$ -failure goal, for some $i \in [0, n]$. Suppose that L_1 is the literal selected

from G_i . If L_1 is a positive literal the proof is the same for (d) in the basis step. We now prove that L_1 cannot be a negative literal. By contradiction, suppose that L_1 is ground negative literal $\neg L$. Since $\neg L$ is a $\Phi[D]$ -failure literal, there is an SLDNF-refutation from $D \cup \{\leftarrow L\}$. Since G contains only positive literals, there is a variant of a clause in U , say C_q , for some $q \in [1, i]$ with $i > 0$, that introduced the negative literal. Suppose that the head of C_q is A . We may construct from S a $\Phi[D]$ -SLDNF-refutation S' from $U[\lambda] \cup \{\leftarrow A\}$ such that $\neg L$ is a $\Phi[D]$ -failure literal for S' . Hence, by Definition 4.3 and step 3 of Procedure 1, and since there is an SLDNF-refutation from $D \cup \{\leftarrow L\}$, we have $\text{success}(L) \in \lambda$. Thus, $L \in U[\lambda]$ and, by Corollary 5.3, $\text{fail}(L) \notin \lambda$. Hence, there is a certified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{\leftarrow L\}$ and the literal L_1 cannot be cancelled in G_i . Contradiction.

(e) Let N be a finitely failed or uncertified $\Phi[\lambda]$ -SLDNF-tree N of rank k from $U[\lambda] \cup \{G\}$. By Definitions 3.4 and 4.1, each branch in N is a failed $\Phi[\lambda]$ -SLDNF-derivation or an uncertified $\Phi[\lambda]$ -SLDNF-refutation, of rank no greater than k . By (b) and (d), N is trivially a finitely failed or uncertified $\Phi[D]$ -SLDNF-tree. ■

The next auxiliary lemma shows that every answer computed by a certified $\Phi[D]$ -SLDNF-refutation does not contradict D .

Lemma 5.5 *Let $\leftarrow Q$ be a query containing positive and negative literals. If R is a certified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{\leftarrow Q\}$, and if α is the answer computed by R for $\leftarrow Q$, there is no finitely failed SLDNF-tree from $D \cup \{\leftarrow Q\alpha\}$.*

PROOF. Let $\leftarrow Q$ be a query containing positive and negative literals. Let R be a $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{\leftarrow Q\}$, and let α be the answer computed by R . Assume that R has length n , the sequence of goals is $G_0 = \leftarrow Q, \dots, G_n$ and the sequence of mgu's is $\theta_1, \dots, \theta_n$. By Proposition 3.6, $Q\alpha$ is ground. Assume that $\leftarrow Q$ is of the form $\leftarrow L_1, \dots, L_m$. Let $i \in [1, m]$. If $L_i\alpha$ is a positive literal A , then there is a literal A' cancelled in a goal of R , say G_k , for some $k \in [0, n)$, such that $A'\theta_{k+1} \dots \theta_n = A$. Moreover, by Definition 4.3, since $\Phi[D](R) = \text{TRUE}$, there is no finitely failed SLDNF-tree from $D \cup \{\leftarrow A\}$. If $L_i\alpha$ is a negative literal $\neg A$, then $\neg A$ is the literal cancelled from some goal of R and, again by Definition 4.3, there is no SLDNF-refutation from $D \cup \{\leftarrow A\}$, which implies that there is no finitely failed SLDNF-tree from $D \cup \{\leftarrow \neg A\}$. Thus, there is no finitely failed SLDNF-tree from $D \cup \{\leftarrow L_i\alpha\}$, for $1 \leq i \leq m$. Hence, there is no finitely failed SLDNF-tree from $D \cup \{\leftarrow Q\alpha\}$. ■

We are now ready to state the major result of this section:

Theorem 5.6 (Correctness of the cooperative interface). *Let C be a dialogue produced by Procedure 1. Then, any log λ in C is sound for D and U with respect to the cooperativeness domain \mathcal{D} .*

PROOF. Straightforward from Lemmas 5.4(c) and 5.5 ■

Therefore, we may assert that Procedure 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.

5.2 Other properties of the interface

In the next sequence of results, let U again be a user model, D be a deductive database, $\leftarrow G$ be a query containing only positive literals, and \mathcal{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 dialogue produced by Procedure 1 and λ be a log in C .

The next lemma, together with Lemma 5.4, establishes a tight relationship between the user's derivations in the presence of a log and the derivations simulated by the interface.

Lemma 5.7

- (a) *If S is a $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$, then S is also a $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$.*
- (b) *If S is a failed $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$, then S is also a failed $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$.*
- (c) *If S is a certified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$, then S is also a certified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$.*
- (d) *If S is an uncertified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$, then S is also an uncertified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$.*
- (e) *If N is a finitely failed or uncertified $\Phi[D]$ -SLDNF-tree from $U[\lambda] \cup \{G\}$, then N is also a finitely failed or uncertified $\Phi[\lambda]$ -SLDNF-tree $U[\lambda] \cup \{G\}$.*

PROOF. The proof will be by induction on the ranks of SLDNF-derivations, certified or uncertified SLDNF-refutations, and finitely failed or uncertified SLDNF-trees.

Basis: Assume that the rank is 0.

(a) Let S be a $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$ of rank 0, that is, no negative literal is selected. Since only positive literals are selected in S , the $\Phi[\lambda]$ and $\Phi[D]$ certification tests have no influence in S . Hence, a $\Phi[D]$ -derivation of rank 0 is also a $\Phi[\lambda]$ -derivation of rank 0.

(b) Let S be a failed $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$ of rank 0. As in (a), S is trivially a failed $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$ of rank 0.

(c) Let S be a certified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$ of rank 0. By (a), S is also a $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$. To prove that S is a certified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$, we will show that the $\Phi[\lambda]$ -certification test succeeds, that is, there are no $\Phi[\lambda]$ -failure goals for S .

Assume that S has length n , the sequence of goals is $G_0 = G, \dots, G_n$ and the sequence of mgu's is $\theta_1, \dots, \theta_n$. Let $i \in [0, n)$. Consider that M is the literal cancelled in G_i . Since S has rank 0, M is a positive literal. Since the $\Phi[D]$ -certification test succeeds, there is no finitely failed SLDNF-tree from $D \cup \{\leftarrow M\theta_{i+1} \dots \theta_n\}$. Thus, by Proposition 5.2, $fail(M\theta_{i+1} \dots \theta_n) \notin \lambda$. Hence, G_i is not a $\Phi[\lambda]$ -failure goal. Therefore, there are no $\Phi[\lambda]$ -failure goals for S .

(d) Let S be an uncertified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$ of rank 0. By (a), S is also a $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$. To prove that S is an uncertified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$, we will show that the $\Phi[\lambda]$ -certification test fails, that is, there is a $\Phi[\lambda]$ -failure goal for S .

Assume that S has length n , the sequence of goals is $G_0 = G, \dots, G_n$, the sequence of variants of clauses in $U[\lambda]$ is C_1, \dots, C_n , and the sequence of mgu's is $\theta_1, \dots, \theta_n$. Let G_i be the $\Phi[D]$ -failure goal, for $i \in [0, n)$. Since S has rank 0, the cancelled literal in G_i is a positive literal, say M , and there is a finitely failed SLDNF-tree from $D \cup \{\leftarrow \neg L\}$, such that $M\theta_{i+1} \dots \theta_n = L$. Assume that A is the head of C_{i+1} . Thus, we may construct from S a $\Phi[D]$ -SLDNF-refutation S' from $U[\lambda] \cup \{\leftarrow A\}$, such that S' is simulated by Procedure 1 in the same iteration where the log λ is generated, and the composition of mgu's is α and $A\alpha = L$. Since there is a finitely failed SLDNF-tree from $D \cup \{\leftarrow \neg L\}$, by Definition 4.3 and step 3 of Procedure 1, $fail(L) \in \lambda$. Hence, G_i is also a $\Phi[\lambda]$ -failure goal.

(e) Let N be a finitely failed or uncertified $\Phi[D]$ -SLDNF-tree N of rank 0 from $U[\lambda] \cup \{G\}$. By Definitions 3.4 and 4.3, each branch in N is a failed $\Phi[D]$ -SLDNF-derivation or an uncertified $\Phi[D]$ -SLDNF-refutation, of rank 0. By (b) and (d), N is trivially a finitely failed or uncertified $\Phi[\lambda]$ -SLDNF-tree.

Induction step: Let $k > 0$. Suppose that the results hold for $j < k$.

(a) Let S be a $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$ of rank k .

Assume that S has length n and the sequence of goals is $G_0 = G, \dots, G_n$. We show that S is also a $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$.

Let $j \in [1, n]$. Suppose that L_1 is the literal cancelled from G_{j-1} . If L_1 is a positive literal the proof is the same as in the basis step. Consider then that the selected literal L_1 is a ground negative literal of the form $\neg L$. Since L_1 is cancelled in S , there is a finitely failed or uncertified $\Phi[D]$ -SLDNF-tree of rank less than k from $U[\lambda] \cup \{\leftarrow L_1\}$. By the induction hypothesis, there is also a finitely failed or uncertified $\Phi[\lambda]$ -SLDNF-tree from $U[\lambda] \cup \{\leftarrow L_1\}$. Thus, L_1 can indeed be cancelled in S .

(b) Let S be a failed $\Phi[D]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$ of rank k . By (a), S is also a $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$. To prove that S is a failed $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$, the literal selected in the last goal of S cannot actually be cancelled.

Assume that S has length n and the sequence of goals is $G_0 = G, \dots, G_n$. Suppose that G_n is a goal of the form $\leftarrow L_1, \dots, L_m$. If L_1 is a positive literal, the proof is the same as for (b) in the basis step. Suppose then that L_1 is a ground negative literal $\neg L$. Since $\neg L$ is not cancelled in G_n , there is a certified $\Phi[D]$ -SLDNF-refutation of rank less than k from $U[\lambda] \cup \{\leftarrow L\}$. Then, by the induction hypothesis, there is also a certified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{\leftarrow L\}$. Hence, L_1 cannot be cancelled in G_n , that is, S is indeed a failed $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$.

(c) Let S be a certified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$ of rank k . By (a), S is also a $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$. The proof that S is a certified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$ is the same as for (c) in the basis step.

(d) Let S be an uncertified $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$ of rank k . By (a), S is also a $\Phi[\lambda]$ -SLDNF-derivation from $U[\lambda] \cup \{G\}$. To prove that S is an uncertified $\Phi[\lambda]$ -SLDNF-refutation from $U[\lambda] \cup \{G\}$, we will show that the $\Phi[\lambda]$ -certification test fails and the $\Phi[\lambda]$ -failure literal is a positive literal.

Assume that S has length n , the sequence of goals is $G_0 = G, \dots, G_n$ and the sequence of variants of clauses in $U[\lambda]$ is C_1, \dots, C_n . Let G_i be the $\Phi[D]$ -failure goal, for $i \in [0, n)$. Suppose that L_1 is the literal cancelled from G_i . If L_1 is a positive literal, the proof is the same as for the basis step. We have to show that L_1 cannot be a negative literal. By contradiction, suppose that L_1 is a ground literal $\neg L$. Since $\neg L$ is a $\Phi[D]$ -failure literal, there is an SLDNF-refutation from $D \cup \{\leftarrow L\}$. Since G contains only positive literals, there is a variant, say C_q , of a clause in U which introduces the negative literal, for some $q \in [1, i]$ with $i > 0$. Suppose that the head of C_q is A . We may construct from S a $\Phi[D]$ -SLDNF-refutation from $U[\lambda] \cup \{\leftarrow A\}$, whose $\Phi[D]$ -failure literal is $\neg L$. Hence, by Definition 4.3 and step 3 of Procedure 1, since there is an SLDNF-refutation from $D \cup \{\leftarrow L\}$, we have $success(L) \in \lambda$. Thus, $L \in U[\lambda]$ and there is a certified $\Phi[D]$ -refutation

from $U[\lambda] \cup \{\leftarrow L\}$. Then, L_1 cannot be cancelled in G_i . Hence, G_i is the last goal of S . Contradiction.

(e) Let N be a finitely failed or uncertified $\Phi[D]$ -SLDNF-tree N of rank k from $U[\lambda] \cup \{G\}$. By Definitions 3.4 and 4.3, each branch in N is a failed $\Phi[D]$ -SLDNF-derivation or an uncertified $\Phi[D]$ -SLDNF-refutation of rank no greater than k . By (b) and (d), N is trivially a finitely failed or uncertified $\Phi[\lambda]$ -SLDNF-tree. ■

6 Conclusions

In this chapter we first described a model for the user's 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 user's 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 are also testing the idea of limiting the domain by capturing in advance the goal(s) that the user wants to achieve [AP80]. In the context of Project NICE, we also explored a simpler approach, based on request modification rules compiled from explicit patterns of user inferences [HCF91a].

References

- [AP80] J. F. Allen and C. R. Perrault. Analyzing intentions in utterances. *Artificial Intelligence*, 15(3):143-178, 1980.
- [BJ86] L. Bolc and M. Jarke, editors. *Cooperative Interfaces to Information Systems*. Springer-Verlag, Berlin, 1986.
- [CCL90] W. Chu, Q. Chen, and R. C. Lee. Cooperative query answering via type abstraction hierachy. In *International Working Conference on Cooperating Knowledge Based Systems*, Univ. Keele, England, 1990.

- [CD89] F. Cuppens and R. Demolombe. Cooperative answering: a methodology to provide intelligent access to databases. In L. Kerschberg, editor, *2nd International Conference on Expert Database Systems*, pages 621–643. Benjamin/Cummings, 1989.
- [CF90] M. A. Casanova and A. L. Furtado. An information system environment based on plan generation. In *International Working Conference on Cooperating Knowledge Based Systems*, Univ. Keele, England, 1990.
- [HCF91a] A. S. Hemerly, M. A. Casanova, and A. L. Furtado. Cooperative behaviour through request modification. In *10th International Conference on the Entity-Relationship Approach*, San Mateo, CA, USA, 1991.
- [HCF91b] A. S. Hemerly, M. A. Casanova, and A. L. Furtado. Exploiting user models to avoid misconstruals. In *1st International Workshop on Nonstandard Queries and Answers Approach*, Toulouse, France, 1991.
- [Kap82] S. J. Kaplan. Cooperative responses from a portable natural language query. *Artificial Intelligence*, 19(2):165–187, 1982.
- [KW89] A. Kobsa and W. Wahlster, editors. *User Models in Dialog Systems*. Springer-Verlag, Berlin, 1989.
- [Llo87] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1987.
- [Mot84] A. Motro. Query generalization: a technique for handling query failure. In *1st International Workshop on Expert Database Systems*, pages 314–325, 1984.
- [Qui89] A. Quilici. Detecting and responding to plan-oriented misconceptions. In A. Kobsa and W. Wahlster, editors, *User Models in Dialog Systems*, chapter 5. Springer-Verlag, Berlin, 1989.
- [Sat87] T. Sato. On Consistency of First Order Logic Programs. Technical Report TR-87-12, Electrotechnical Laboratory, Ibaraki, Japan, 1987.
- [Web86] B. L. Webber. Questions, answers and responses: interacting with knowledge base systems. In M. L. Brodie and J. Mylopoulos, editors, *On Knowledge Base Management Systems*. Springer-Verlag, Berlin, 1986.