Consistency Verification of Deductive Database Schemes*

Francisco Marqués and Juan C. Casamayor

Dept. Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n. 46071 Valencia (Spain)
{pacom,carlos}@dsic.upv.es

Abstract

A deductive database scheme consists of a set of base predicate schemes and a set of possibly non-Horn clauses. A database can be considered as a particular instance of a given scheme. Clearly, a property that must be held by any scheme is logical consistency. We also propose other related properties (non-trivial consistency, strong consistency and P-consistency). These properties are requirements which are reasonable and meaningful in many practical cases. For verifying the various forms of consistency, we discuss a method whose centerpiece is a theorem prover which is an instance of a linear-resolution-based paradigm for first-order theories, called SL* resolution.

INTRODUCTION

A *first-order theory*, from now on a f.o. theory, is a set of first-order clauses. Each of these clauses is a non-empty disjunction of literals. The format representation that we have chosen for clauses is in implication form, that is: a clause is represented as a head (the consequent) and a body (the antecedent), where the head is a possibly empty disjunction of atoms, and the body is a possibly empty conjunction of atoms. Thus, a f.o. theory can be understood as a generalization of a Horn theory. The appearance of the disjunction in the head of the clauses is necessary whenever disjunctive knowledge has to be expressed. However, the expressiveness gained by introducing disjunction involves an increase of the complexity of the procedures that deal with theories of this kind.

* This work was supported by a CICYT grant nº TIC93-0473

A *first-order deductive database*, abbreviated a DDB, is a first-order theory, that generalizes the classical concept of a deductive (definite) database. A DDB can be considered as a particular instance of a *first-order database scheme* (DDBS). Essentially, a DDBS consists of a set of base predicate schemes and a f.o. theory. Each base predicate scheme represents the format of possible unit base knowledge which a DDB, instance of the DDBS, may contain. The f.o. theory represents the deductive knowledge (clauses with a non-empty head) and the integrity theory (clauses with an empty head, also called *denials*) associated to the DDBS. For further details on deductive databases, see [Ko2], [LMR], [Ul].

As a f.o. theory may be inconsistent (unsatisfiable), the next question immediately follows: Given a DDBS, is this scheme consistent? That is, is the f.o. theory of this scheme consistent? Moreover, if the given scheme is consistent, are there meaningful databases, instances of this scheme?. These are significant questions because, in the first place, if a DDBS is not consistent, then you can derive everything from the DDBs instances of this scheme. And, in second place, if the DDBS is consistent, then the unique consistent instance of it might not be meaningful (for example, there are DDBSs whose unique consistent instance does not have base information at all). We call schemes of this kind *trivially consistent* ones, and hence, *non-trivially consistent* schemes are those in which this undesirable property does not hold. Other scheme properties related to the questions just stated and studied throughout this paper, are *strong consistency* and *P-consistency*. For a strongly consistent scheme, it is possible to ensure the existence of a DDB, instance of it, with base information for each base predicate scheme. P-consistency can be viewed as an intermediate property between non-trivial and strong consistency.

In this paper, we propose a method for verifying the accomplishment of these properties by a given DDBS (which is an extension of the mechanism presented in [CDM]). Given a property to be checked, this method consists of obtaining an "equivalent" f.o. theory from the scheme (equivalent wrt the property to be verified), and studying the (logical) consistency of this f.o. theory, so that if the f.o. theory is consistent, then the scheme accomplishes the property. In order to make this consistency study of the f.o. theory, we use a theorem prover that is an instance of a linear-resolution-based reasoning paradigm called SL*. This paradigm, which was presented in [DC], is the basis for complete proof procedures that need only a small amount of ancestor resolution and no factoring. In particular, there exists an instance of this paradigm that coincides with SLD resolution for Horn theories.

Unfortunately, the study of the consistency of a f.o. theory is troublesome, since the problem of generally determining whether a first-order theory is consistent or not, is undecidable. However, we should not forget that there exist particular and interesting cases where this question still remains decidable. For example, in the propositional case the problem is known to be np-complete [GJ], and in the datalog case (i.e. there are no function symbols in the theory), the problem is decidable.

The paper is organized as follows: in section 1, we define the main concepts and properties we deal with; in section 2, we present a two-phase method for studying these properties; in section 3, we present the SL* paradigm, focusing the discussion on two procedures (theorem provers), which are instances of it; in section 4, we present the method that incorporates one of these procedures and we show the main results of our work; in section 5, we give a complete example using our method, and finally, we draw some conclusions and propose possible future work.

1 DEFINITIONS

In this section, our aim is to introduce and discuss several definitions of the main concepts and properties presented in our work.

Definition 1

A *f.o. theory* is a set of clauses, each one of which is of the form $A_1 \vee \ldots \vee A_k \leftarrow B_1 \wedge \ldots \wedge B_l$ ($k \geq 0$, $l \geq 0$, $k+l \geq 1$), where $A_1, \ldots, A_k$ and $B_1, \ldots, B_l$ are atoms.

Definition 2

A *deductive database scheme* is a pair (BP, Cl), where:

+ BP is a set $\{bp_1, \ldots, bp_m\}$ of base predicate schemes. Each base predicate scheme bp_i is of the form $bp(x_1, \ldots, x_j)$, where $x_1, \ldots, x_j$ are variables. (For simplicity, we assume a universal domain over which variables range, rather than using typed attribute variables).

+ Cl is a f.o. theory, that is: a set $\{c_1, \ldots, c_n\}$ of clauses, of the form $A_1 \vee \ldots \vee A_k \leftarrow B_1 \wedge \ldots \wedge B_l$ ($k \geq 0$, $l \geq 0$, $k+l \geq 1$) such that the predicate of each atom A_j is a derived predicate, i.e. there is no atom A_j whose predicate is in one of the base predicate schemes.

Note that this definition introduces a clear division among the predicates (base and derived) of a deductive database scheme. The variables that appear in the base predicate scheme indicate that knowledge about which ground atoms are true for each base predicate is not necessarily known at specification time. Different sets of ground atoms for these base predicates will correspond to different databases of the same DDBS. This concept of a deductive database is defined below.

Definition 3

Let S be a deductive database scheme of the form (BP, Cl). A *deductive database* of this scheme S is a pair (F, DCl) such that each element A in F is a ground instance of some bp_i in BP, and DCl is a set of clauses such that $Cl \subseteq DCl$ and the predicate of each atom in the head of every clause in DCl is a derived predicate. Atoms in F are called *facts*.

Because of the possible presence of negative clauses in the set Cl of a DDBS, Cl may be inconsistent. In the case of inconsistency, no database of this scheme has a consistent set of clauses

DCl. Thus, a method for verifying the consistency (satisfiability) of Cl is needed. Below, we give a formal definition of DDBS consistency.

Definition 4

Let S=(BP, Cl) be a deductive database scheme. S is a *consistent scheme* iff Cl is consistent. We say that S is *inconsistent* iff S is not consistent.

On the other hand, there may exist consistent schemes for which the unique DDBs that satisfy them are the ones that do not contain any fact at all. For example, the scheme ({p(x)}, {←p(x)}) is consistent; however, the unique DDBs that satisfy this scheme are of the form (∅, DCl), where DCl contains ←p(x). From a practical point of view, we think that schemes of this kind are not worthwhile, although they are consistent. The schemes that do not have this undesirable characteristic are called *non-trivially consistent schemes*. The definition below formalizes this concept.

Definition 5

Let S=(BP, Cl) be a deductive database scheme such that BP is $\{bp_1, \ldots, bp_n\}$. S is a *non-trivially consistent scheme* iff $\{\exists(bp_1) \vee \ldots \vee \exists(bp_n)\} \cup Cl$ is a consistent set of formulas. S is a *trivially consistent scheme* iff S is a consistent scheme and is not a non-trivially consistent scheme.

Note that if a scheme S is non-trivially consistent, then S is consistent. According to this definition, if a DDBS is non-trivially consistent, then the possibility of the existence of a DDB that satisfies this scheme with a non-empty set of facts F is ensured.

Example 1

Let S=(BP, Cl) be a deductive database scheme such that

BP = {p(x)}

Cl = {q(x) ∨ r(x) ← p(x),
t(x) ← r(x),
← q(a) ∧ p(a),
← t(a) ∧ p(a) },

where *a* is a constant and *x* is a variable. Clearly, S is a non-trivially consistent scheme. Therefore, the existence of a DDB for this scheme is ensured (e.g. the database formed by (F, Cl), where F is {p(b), p(c)}, and *b*, *c* are constants).

On the other hand, the property of strong consistency can be understood as a strengthening of non-trivial consistency for ensuring the existence of a database that contains, at least, one fact for each base predicate scheme of its DDBS. The following definition formalizes this property.

Definition 6

Let S=(BP, Cl) be a deductive database scheme such that BP is $\{bp_1, \ldots, bp_n\}$. S is a *strongly consistent scheme* iff $\{\overline{\exists}(bp_1), \ldots, \overline{\exists}(bp_n)\} \cup Cl$ is a consistent set of formulas.

Finally, the accomplishment of the P-consistency property by a given scheme with base predicate scheme set BP, ensures the existence of a database that contains, at least, one fact for each base predicate scheme in P, where P is a non-empty subset of BP. Clearly, if P is equal to BP, then P-consistency coincides with strong consistency.

Definition 7

Let S=(BP, Cl) be a deductive database scheme such that BP is $\{bp_1, \ldots, bp_n\}$ and let P be a non-empty subset of BP. S is a *P-consistent scheme* iff $\bigcup_i\{\overline{\exists}(bp_i): bp_i \in P\} \cup Cl$ is a consistent set of formulas.

We think that DDBSs, in which any of these properties do not hold, may be easily specified, particularly, when these schemes contain a large amount of unstructured knowledge. Our aim in introducing these properties is to give the designer some concepts to be able to check that his specification is representing his intended interpretation of the real world correctly. Moreover, the task of verifying whether a scheme accomplishes some given property may be extremely hard, if it is done without the help of mechanical tools. In the following sections, we propose a method for this.

Example 2

Let S=(BP, Cl) be a deductive database scheme such that

$BP = \{p(x), q(x), r(x)\}$

$Cl = \{d(x,a) \vee d(y,b) \leftarrow p(x) \wedge r(y)$
$g(x) \vee s(x) \leftarrow p(x)$
$u(x) \leftarrow g(x)$
$s(x) \leftarrow r(x) \wedge q(y)$
$g(x) \leftarrow d(x,a) \wedge s(x)$
$\leftarrow d(x,a) \wedge u(x)$
$\leftarrow d(x,b) \wedge s(x)\}$,

where *a*, *b* are constants and *x*, *y* are variables. For this example Cl is clearly a consistent theory, so S is also a consistent scheme; but it is not a strongly consistent scheme because the theory $\{\exists x\, p(x), \exists y\, q(y), \exists z\, r(z)\} \cup Cl$ is unsatisfiable; S is also a non-trivially consistent scheme because the theory $\{\exists x\, p(x) \vee \exists y\, q(y) \vee \exists z\, r(z)\} \cup Cl$ is satisfiable; finally there exist subsets P of BP such that for these subsets the scheme is P-consistent, for example let P be the set $\{p(x), q(x)\}$, then the theory $\{\exists x\, p(x), \exists y\, q(y)\} \cup Cl$ is also satisfiable. The other two subsets of BP for which the DDBS is also a P-consistent scheme are: $\{p(x), r(x)\}$ and $\{q(x), r(x)\}$.

The relationships among the four properties defined above are stated in the following theorem.

Theorem 1

Let S = (BP, Cl) be a DDBS then

a) S is a consistent scheme if S is a non-trivially consistent scheme.

b) S is a non-trivially consistent scheme if S is a strongly consistent scheme.

c) S is a non-trivially consistent scheme if there exists a non-empty set $P \subseteq BP$ such that S is a P-consistent scheme.

Proof:

Let S be the scheme (BP, Cl), where BP is $\{bp_1, \ldots, bp_n\}$.

The proof of a) is straightforward. If S is non-trivially consistent, then the set of formulas $\{\overline{\exists}(bp_1) \vee \ldots \vee \overline{\exists}(bp_n)\} \cup Cl$ is consistent. Therefore, Cl is also consistent. Hence, S is a consistent scheme by definition.

The proof of b) is as follows: If S is strongly consistent, then the set of formulas $\{\overline{\exists}(bp_1), \ldots, \overline{\exists}(bp_n)\} \cup Cl$ is consistent. Moreover, any model of $\{\overline{\exists}(bp_1), \ldots, \overline{\exists}(bp_n)\} \cup Cl$ is also a model of $\{\overline{\exists}(bp_1) \vee \ldots \vee \overline{\exists}(bp_n)\} \cup Cl$. Therefore, by definition, S is a non-trivially consistent scheme. A similar proof can be formulated for point c).

2 Consistency Verification of Deductive Database Schemes

In this section, we describe a method for verifying the defined properties of deductive database schemes. Roughly, our method consists of two phases: first, generating a first order theory, in accordance with the property to be verified, from the deductive database scheme; and second, checking the logical (in)consistency of this generated first order theory by using an appropriate theorem prover. Below, we present these two phases with their main results.

2.1 Generation Phase

This first phase of the method generates a f.o. theory from the given DDBS. This generated theory is related to the property to be verified, so that the DDBS accomplishes the property iff this theory is consistent. Thus, the generated theory will be different according to the corresponding property. We give a definition describing the generation phase of the method below.

Definition 8 (Generation phase)

Let S be a deductive database scheme of the form (BP, Cl), where BP is $\{bp_1, ..., bp_n\}$. The f.o. theory T obtained by the generation phase of the method corresponding to the property to be verified is described below:

a) Consistency verification: T is equal to Cl.

b) Non-trivial consistency verification: T is $\{sk(\overline{\exists} bp_1) \vee ... \vee sk(\overline{\exists} bp_n)\} \cup Cl$.

c) Strong consistency verification: T is $\{sk(\overline{\exists} bp_1), ..., sk(\overline{\exists} bp_n)\} \cup Cl$.

d) P-consistency verification: T is $\bigcup_j \{sk(\overline{\exists} bp_j): bp_j \in P\} \cup Cl$, where $P \subseteq BP$.

where $sk(\overline{\exists} bp_i)$ denotes the skolemized base predicate scheme bp_i.

Example 3

Let S be the DDBS of example 2. The properties to be verified are non-trivial consistency and P-consistency, {p(x), q(x)} being the set of base predicate schemes P. Then, the generated theories T_1 and T_2, respectively, are the following:

- $T_1 = \{p(sk_1) \vee q(sk_2) \vee r(sk_3)\} \cup Cl$
- $T_2 = \{p(sk_1), q(sk_2)\} \cup Cl$

where sk_1, sk_2 and sk_3 denote three skolem constants.

The next theorem states the equivalence between the accomplishment of some property by a given scheme and the consistency of the corresponding theory obtained by the generation phase. For simplicity, a DDBS property is one of the previously defined properties in section 1 (i.e. consistency, non-trivial consistency, strong consistency, or P-consistency)

Theorem 2

Let S be a deductive database scheme of the form (BP, Cl), V be a DDBS property and let T be the generated theory corresponding to the property V to be verified. S satisfies the property V iff T is a consistent f.o. theory.

Proof:

Let S be of the form (BP, Cl) such that BP is $\{bp_1, ..., bp_n\}$.

If P is consistency, then the proof is straightforward.

If P is trivial consistency, then the proof is as follows: By definition of non-trivial consistency, $\{\overline{\exists}(bp_1) \vee ... \vee \overline{\exists}(bp_n)\} \cup Cl$ is a consistent set of formulas. It is well-known that skolemization preserves consistency. Therefore, $\{\overline{\exists}(bp_1) \vee ... \vee \overline{\exists}(bp_n)\} \cup Cl$ is consistent iff the corresponding skolemized f.o. theory $\{sk(\overline{\exists} bp_i) \vee ... \vee \overline{\exists}(bp_n)\} \cup Cl$ is consistent. Since this theory is T, the result holds. Similar proofs can be formulated for the property V being either strong consistency or P-consistency.

2.2 (IN)CONSISTENCY CHECKING PHASE

The second phase of the method consists of checking the (in)consistency of the theory generated by the previous first phase. Clearly, some mechanical tool is needed for achieving this verification. It is well-known that consistency is an undecidable property for f.o. theories, although there exist complete procedures for checking the inconsistency of an unsatisfiable f.o. theory. Besides, there are classes of f.o. theories (propositional, function-free, etc.) where consistency is decidable.

Clearly, a very suitable tool for studying the (in)consistency of f.o. theories is a theorem prover. Probably, the most well-known theorem provers are based on the Resolution Principle [Ro]. The basis of resolution-based theorem proving can be found in [Lo1] [CL]. Soundness and completeness are two essential properties which must be accomplished by every theorem prover, ensuring that a f.o. theory is inconsistent iff this fact is detected by the theorem prover. Additionally, the implementation of the corresponding theorem prover must preserve these properties, taking into account problems like sound unification, complete search, etc.

In the next section, we present a linear-resolution-based paradigm, called SL*, and two instances of it that are sound and complete proof procedures for checking the inconsistency of f.o. theories. In section 4) we state the results of the whole method for verifying the properties of a DDBS, that includes the use of these procedures.

3 SL* RESOLUTION

In this section, we present a linear-resolution-based paradigm for monotonic reasoning in first order theories, called SL* resolution [DC]. Instances of this paradigm correspond to different procedures that are adequate to various purposes, namely, theorem proving (inconsistency checking) and problem solving (query answering). Some of these procedures are suited for certain classes of theories (non-Horn, near-Horn, Horn, range-restricted theories) and for obtaining certain classes of solutions (definite or indefinite). In particular, an instance of SL* for Horn theories is identical to SLD resolution [KK] [Ko1]. All these procedures are sound and complete for the corresponding purpose for which they were tailored in the corresponding class of theories which they deal with.

Linear Resolution can be understood as an extension of Input Resolution in which, each predecessor clause (an ancestor) in the derivation is also considered as a possible input clause. Roughly, SL* refines Linear Resolution taking into account only previously selected literals as ancestors. This is the minimum extension to Input Resolution needed for obtaining a complete procedure for f.o. theories [Lo1],[St1]. Essentially, two different inference steps can be

distinguished in SL*: First, a binary resolution step on the current clause in the derivation and an input clause from the current theory (the initial theory and the set of current ancestors), and second, an application of the *splitting rule* [CL] to the current clause on the selected literal.

Two different cases may occur in the first step: First, the input clause is a fresh variant of a clause in the initial theory, then, this step corresponds to an input resolution step (*extension*, in other terminologies); second, the input clause is an ancestor, then, this step corresponds to an ancestor resolution step (sometimes called *reduction*, and corresponding to an application of the principle of *reductio ad absurdum*).

The application of the splitting rule in the second step can be explained as follows (we consider the ground case for simplicity): Suppose that the (in)consistency of a theory T and a ground clause C is required to be checked. C is of the form $L_1 \vee \ldots \vee L_n$ and a literal L_i is selected in C. Then, the problem of checking the (in)consistency of $T \cup \{C\}$ can be reduced to checking the (in)consistency of $T \cup \{L_i\}$ and of $T \cup \{L_1 \vee \ldots \vee L_{i-1} \vee L_{i+1} \vee \ldots \vee L_n\}$. In the non-ground case, this application must be done taking into account the shared variables in the two parts of the split clause. In SL* resolution, the checking of the (in)consistency of $T \cup \{L_i\}$ is done by a subsidiary derivation of lower rank, adding L_i to the set of ancestors. These literals, which are the root of subsidiary derivations, are called *ancestors*. In order to guarantee completeness, this second step has to be applied whenever the selected literal belongs to a certain class. This class is formalized by defining the concept of *ancestor choice*.

Definition 9

Given a f.o. theory T, an *ancestor choice* M for T is a (possibly empty) subset of the set of all literals L of the underlying language.

Actually, an ancestor choice M allows us to define the condition for a selected literal being an ancestor. This condition is formalized in the definition of SL* refutation, that is given below.

For convenience, we first define: For a set of clauses S and a substitution θ, let $S\theta$ denote the set of clauses obtained by applying θ to each variable in each clause of S.

Definition 10

Let T be a f.o. theory, C a clause and Anc a set of clauses. An *SL* refutation from* C *in* T *using* Anc, and its *rank*, are defined as follows: The refutation consists of a sequence of clauses $C_0,\ldots, C_n$, a sequence $\theta_1,\ldots, \theta_n$ of substitutions and a sequence of sets of clauses $Anc_0,\ldots, Anc_n$ called *sets of ancestors* ($n \geq 0$), such that $C_0=C$, $C_n=[]$ (the empty clause), $Anc_0=Anc$, $Anc_i=Anc\theta_1\ldots\theta_i$ ($1 \leq i \leq n$) and, for each $i<n$, an atom A_i is selected in C_i. Moreover, the four points below hold.

- Suppose M is a ancestor choice for T. For each i ($0 \leq i < n$), A_i is called an *ancestor* if A_i is selected in the head (resp. the body) of C_i, $A_i \in M$ (resp. $\neg A_i \in M$), and $A_i \notin T \cup Anc_i$ (resp. $\leftarrow A_i \notin T \cup Anc_i$).

- If, for each i ($0 \leq i < n$), A_i is not an ancestor, then the rank of the refutation is 0; otherwise, it is k, for some sufficiently large $k > 0$. (A more elaborate definition would induct on the rank, as for SLDNF in [Ll]).

- If A_i is not an ancestor, then there is a clause C' such that C' is either a fresh variant of a clause in T, or $C' \in Anc_i$, and C_{i+1} is a resolvent of C_i and C' on A_i and some atom A in C', using θ_{i+1} which is a mgu of A_i and A.

- If A_i is an ancestor and is selected in the head (resp. the body) of C_i, then there is an SL* refutation from $A_i \leftarrow$ in T using $Anc_i \cup \{A_i \leftarrow\}$ (resp. from $\leftarrow A_i$ in T using $Anc_i \cup \{\leftarrow A_i\}$), of rank less than k, with computed substitution θ_{i+1}, and C_{i+1} is the resolvent of C_i and $\leftarrow A_i\theta_{i+1}$ (resp. of C_i and $A_i\theta_{i+1} \leftarrow$) on A_i.

The composition $\theta_1 \ldots \theta_n$ is called the *computed substitution* of the refutation.

For brevity, we omit definitions of "finitely failed SL* tree" (i.e. each attempt of finding an SL* refutation in the search space terminates with failure), "SL* derivation" and "SL* tree".

The fourth point in the definition corresponds to the application of the splitting rule as we explained above. Intuitively speaking, when this point is applied, the problem of checking the inconsistency of C_i with the current theory is reduced to (or *split*), first, checking the inconsistency of the selected literal with the current theory by a subsidiary refutation of lower rank from this literal, and second, checking the inconsistency of C_{i+1} with the current theory. Note that, in the subsidiary refutation, the selected literal is incorporated to the set of ancestors, avoiding that fresh variants of this literal may be used as input clauses. This ensures that the negation of this selected literal instantiated by the computed substitution is a logical consequence of the current theory. Also, note that C_{i+1} is obtained by taking care of the computed substitution of the lower rank refutation for the literal. These facts guarantee that the application of the splitting rule is adequate.

As we mentioned before, there are distinct instances of the SL* resolution paradigm which yield different proof procedures, each of them suited to various purposes. For example, if we choose the following instance of the paradigm: the theory T is definite, the top clause C is a denial, Anc (the set of ancestors) is empty, and M (the ancestor choice) is also empty, then we obtain SLD resolution. Essentially, different instances can be obtained by deciding: a) the characteristics of the initial theory T; b) whether the top clause C belongs to T or it is initially included in the set of ancestors Anc; and c) the ancestor choice M which will control the application of the subsidiary computations. The decisions a) and b) will result in different instances of the paradigm, each of

them suited for consistency checking (theorem proving) or problem solving (query answering) in theories of a certain class. The decision c) can be essentially understood as a control mechanism, since subsidiary ranks are only taken from ancestors, as defined above.

Some ancestor choices are of particular interest. The empty ancestor choice, that implies that no subsidiary ranks will be taken at all (simulating Input Resolution). The positive ancestor choice POS, that contains all atoms; therefore, subsidiary ranks will only be taken from the positive literals in the current clause of the derivation. Finally the total ancestor choice ALL, that contains all literals; in this case, subsidiary ranks will be taken from every literal selected in the current clause of the derivation. Clearly, not every ancestor choice ensures the completeness of the corresponding procedure (in particular, the empty ancestor choice results in a procedure that corresponds to Input Resolution, that is well-known to be incomplete when dealing with f.o. theories). For ensuring completeness, the ancestor choice must accomplish that, for each possible atom, either it or its negation (or both) are in the ancestor choice. We call an ancestor choice of this type a *complete ancestor choice*. On the other hand, there are *ground* ancestor choices (containing only ground literals) that are adequate for dealing with range-restricted theories.

In the next section, we present two complete proof procedures, instances of SL* resolution, for theorem proving (or, inconsistency checking) in f.o. theories, called SLT-ALL (with the total ancestor choice ALL) and SLT-POS resolution (with the positive ancestor choice POS). In both proof procedures, the top root C is included in the set of clauses, allowing the procedure to use a fresh variant of the initial clause in the derivation, while Anc (the set of ancestors) is initially empty. A complete proof procedure for theorem proving in range-restricted theories, called SLC resolution, was presented in [De] [DC]. This procedure is an instance of SL* which is defined by using a ground ancestor choice.

3.1 SLT-ALL RESOLUTION AND SLT-POS RESOLUTION

SLT-ALL resolution is a complete proof procedure for theorem proving that corresponds to an instance of SL*, in which the ancestor choice M is the total ancestor choice. As we mentioned before, in this case, subsidiary ranks will be taken from every literal selected in the current clause of the derivation. We define an SLT-ALL refutation as follows:

Definition 11

Let T be a theory and C a clause. Then, an *SLT-ALL refutation* from C in T is an SL* refutation from C in $T \cup \{C\}$ using $\varnothing$, such that the considered ancestor choice M for T contains all literals of the underlying language.

Example 4

Let T be the following f.o. theory:

T= { p(x) v q(x), 1
q(x)←p(x), 2
p(a)←q(a) } 3

where *a* is a constant and *x* is a variable. The clause C is ←p(x)∧q(x). Clearly, T ∪ {C} is inconsistent. In the figure below, an SLT-ALL refutation from C in T is shown. The selected atoms are underlined if necessary, the input clause used in each step is shown and the different boxes contain the refutations of higher-to-lower ranks in a nested manner.

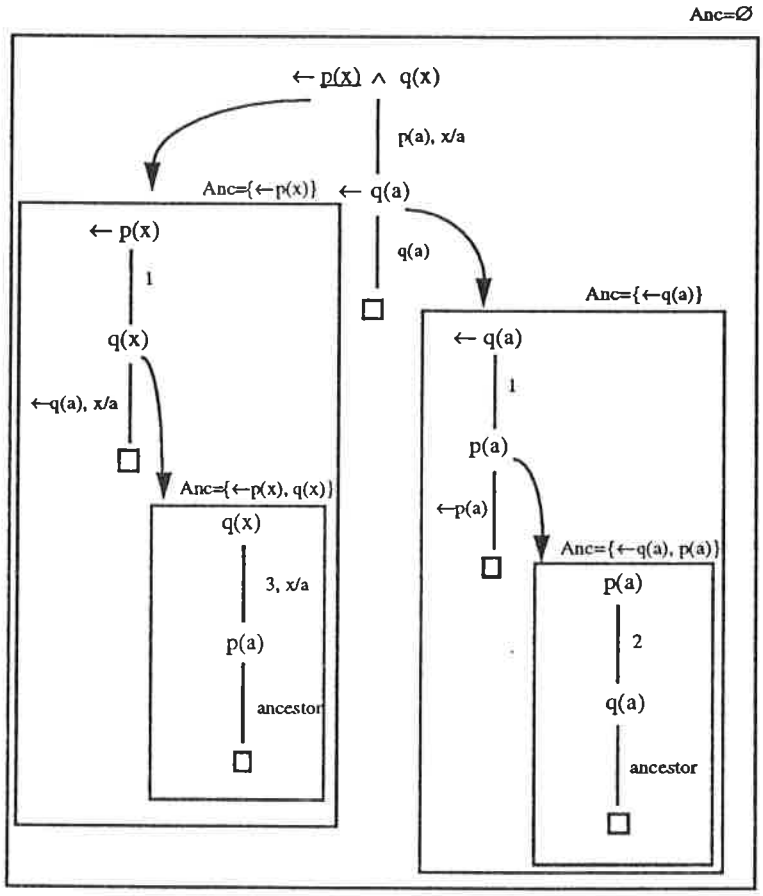

Fig. 1: SLT-ALL refutation for example 4

The figure above shows an SLT-ALL refutation from ←p(x)∧q(x) in T ∪{←p(x)∧q(x)} of rank 2, with computed substitution {x/a}. An example of the application of subsidiary computations occurs in the first step of this refutation. In this step, p(x) is selected in the body of the top clause and, since ←p(x) is an ancestor, a subsidiary derivation from ←p(x) in T ∪{←p(x)∧q(x)} using {←p(x)} as set of ancestors starts from it. This derivation becomes a refutation of rank 1 with

computed substitution {x/a}, showing that $T \cup \{\leftarrow p(x) \wedge q(x)\} \models p(a)$. Therefore, p(a) can be used as an input clause in the higher derivation. A similar explanation can be done, for the other subsidiary refutations.

It can be shown that SLT-ALL resolution coincides with other well-known theorem provers for f.o. theories in the literature, in some sense. All of these theorem provers are based on the Model Elimination procedure, or on reformulations of it, as the MESON format. References for these procedures can be found in [Lo1]. There are also some recently proposed theorem provers like PTTP [St1][St2] and Plaisted's proposal [Pl], that can be understood as implementations of the ME procedure. All of these procedures are based on a model elimination (generation) refutation paradigm that consists of proving the inconsistency of a set of clauses by showing that no model can be built for this set. Obviously, SLT-ALL can also be understood in this way.

SLT-POS resolution is also a complete proof procedure for theorem proving that corresponds to an instance of SL*, in which the ancestor choice M is the positive ancestor choice. Therefore, subsidiary ranks will be taken only from the positive literals selected in the current clause of the derivation. We define an SLT-POS refutation as follows:

Definition 12

Let T be a theory and C a clause. Then, an *SLT-POS refutation* from C in T is an SL* refutation from C in $T \cup \{C\}$ using $\varnothing$, such that the considered ancestor choice M for T contains all atoms of the underlying language.

Example 5

Let T be the following f.o. theory:

$T = \{\ p(x) \vee q(x), \quad 1$
$\quad q(x) \leftarrow p(x), \quad 2$
$\quad p(a) \leftarrow q(a)\ \} \quad 3$

where *a* is a constant and *x* is a variable. The clause C is $\leftarrow p(x) \wedge q(x)$. Note that this example is identical to example 4.

In figure 2, an SLT-POS refutation from C in T is shown. This figure shows an SLT-POS refutation from $\leftarrow p(x) \wedge q(x)$ in $T \cup \{\leftarrow p(x) \wedge q(x)\}$ of rank 2, with computed substitution {x/a}. As in example 4, subsidiary refutations can be explained analogously. The main difference between SLT-POS and SLT-ALL resolution can be observed in the higher refutation. In the SLT-POS refutation, the first two steps correspond to input resolution steps, since the selected atoms were chosen in the body. That is, SLT-POS resolution does not apply subsidiary computations until an atom is selected in the head. Thus, SLT-POS behaves like SLD resolution if the given theory is Horn and the initial clause is a denial. This fact cannot be stated for SLT-ALL resolution because of its ranking policy.

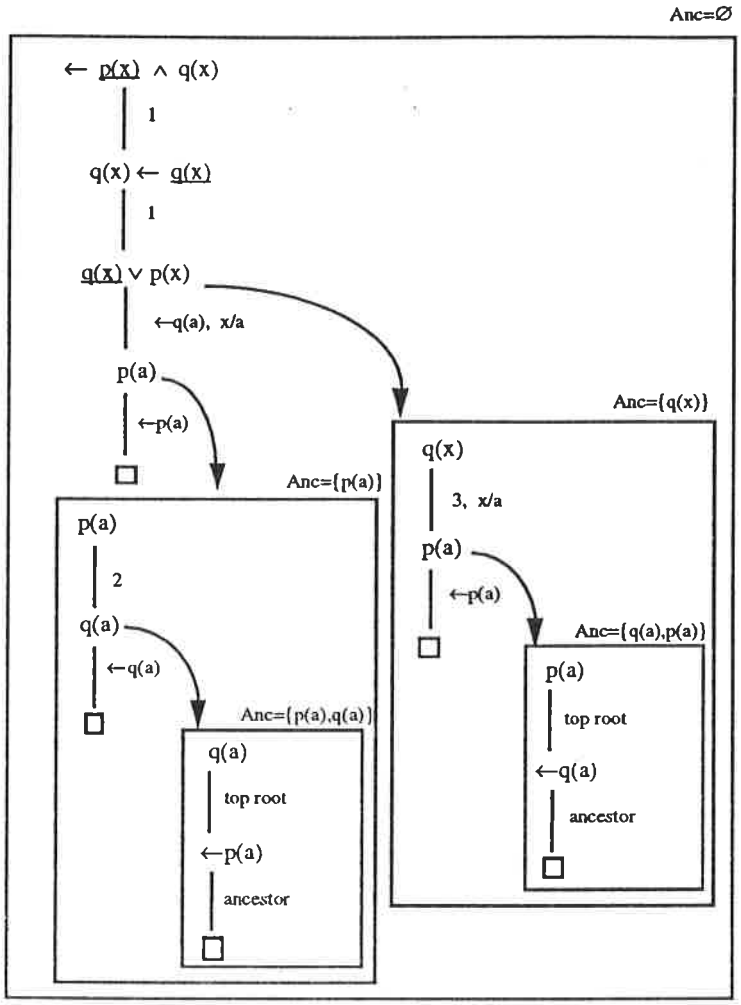

Fig. 2: SLT-POS refutation for example 5

The main justifications of SLT-POS resolution are its completeness for f.o. theories and its adequacy for near-Horn theories [Lo2]. A near-Horn theory is a f.o. theory that contains few non-Horn clauses. In the field of Databases, most of the problems can be represented by using Horn clauses, and occasionally, only a few non-Horn clauses may be needed. The reason for claiming the adequacy of SLT-POS resolution for theories of this class is that SLT-POS behaves similarly to SLD resolution (inheriting its characteristics) while non-Horn clauses are not used in the computation. Since the presence of non-Horn clauses is small, this situation occurs frequently. On the other hand, even if non-Horn clauses are used in the computation, the amount of ancestors needed for ensuring completeness is usually less than the amount needed in other theorem provers (for example, in SLT-ALL resolution). We should note that there are problem classes (mathematical theorems, logic puzzles, etc.) not suitable for SLT-POS resolution, other theorem provers (like SLT-ALL resolution) being more adequate.

The following theorem states the main results of SLT-ALL and SLT-POS resolution. In the theorem, any reference to SLT-POS can be replaced by SLT-ALL.

Theorem 3

Let T be a f.o theory and C a clause.

a) If there is an SLT-POS refutation from C in T, then T ∪ {C} is inconsistent.

b) If T is inconsistent, then there is a denial in T such that there is an SLT-POS refutation from that denial in T.

c) If T is a consistent theory and there is a finitely failed SLT-POS tree from C in T, then T∪{C} is consistent.

d) If there is a finitely failed SLT-POS tree from each denial in T, then T is consistent.

Proof: the proofs of these results can be found in [Ca].

4. USING SLT-POS RESOLUTION IN THE CONSISTENCY VERIFICATION METHOD

Clearly, the SLT-ALL or SLT-POS procedure could be incorporated into the method for verifying the properties of the DDBSs, defined in section 2, in order to check the (in)consistency of the f.o. theory generated by the first phase of the method (the generation phase). However, since the usual form of DDBs are near-Horn, we propose the use of SLT-POS because of the reasons addressed above.

On the other hand, since the f.o. theory T generated from the DDBS by the generation phase of the method may not be minimally unsatisfiable, the clause of the theory T to be used as initial root in the SLT-POS derivation is not known. For this reason, we apply a transformation to the theory T, called *focusing-on-denials phase* (briefly, *f-on-d*), to obtain a f.o. theory, denoted by f-on-d(T), such that there exists a known clause to be used as the top root. This transformation is defined below.

Definition 13

Let T be a f.o. theory. The theory *f-on-d(T)* is obtained by replacing each denial of form ←B in T by inconsistent←B, where inconsistent is a 0-ary predicate that does not occur in T.

The following theorem shows that the above transformation preserves the (in)consistency of the theory. Besides, it also shows that the ←inconsistent clause can be used as the top root by SLT-POS resolution.

Theorem 4

Let T be a f.o. theory. The following three points hold:

a) T is inconsistent iff f-on-d(T) ∪ {←inconsistent} is inconsistent

b) T is inconsistent iff there exists an SLT-POS refutation from ←inconsistent in f-on-d(T).

c) If there is a finitely failed SLT-POS tree from ←inconsistent in f-on-d(T), then T is consistent

Proof: The proof of point a) is straightforward.

The proof of point b) is as follows: T is inconsistent iff f-on-d(T) $\cup$ {$\leftarrow$inconsistent} is inconsistent, by point a). By results from theorem 3 a) and 3 b), f-on-d(T) $\cup$ {$\leftarrow$inconsistent} is inconsistent iff there exists an SLT-POS refutation from $\leftarrow$inconsistent in f-on-d(T).

The proof of c) is straightforward from point a) and theorem 3 d).

Now, we state the properties of the whole consistency verification method using SLT-POS resolution, as a corollary of the previous results. For simplicity, a DDBS property is one of the previously defined properties in section 1 (i.e. consistency, non-trivial consistency, strong consistency, or P-consistency)

Corollary 1

Let S=(BP,Cl) be a DDBS, V be a DDBS property, and T be the theory obtained by the generation phase of the method corresponding to the property V to be verified. Then the following two points hold:

a) There is an SLT-POS refutation from $\leftarrow$inconsistent in f-on-d(T) iff S does not satisfy the property V.

b) If there exists a finitely failed SLT-POS tree from $\leftarrow$inconsistent in f-on-d(T), then S satisfies the property V.

Clearly, given a scheme S, it may accomplish only some, if any, of the properties defined in section 1. The above corollary and theorem 1 allow us to infer the characteristics of a given scheme from the outcome of the application of the method. For example, if we know that there exists a subset P of BP such that the scheme S is P-consistent by applying the method, then we can deduce that S is a non-trivially consistent scheme, and therefore, S is also consistent.

5. AN EXAMPLE

In this section, we show an example of a DDBS together with a complete description of the application of our method for studying the verification of the defined properties by the scheme. This example is in the medical diagnosis field.

Deductive database scheme MD = (BP, Cl):
Base predicate schemes BP= {Chest-Pain(x)}
Clauses Cl={Disease(x, 'Heart-disease')$\vee$Disease(x, 'Lung-disease')$\leftarrow$Chest-Pain(x),
Go-to-doctor(x) $\vee$ Do-Sport(x) $\leftarrow$ Chest-Pain(x),
Untroubled(x) $\leftarrow$ Go-to-doctor(x),
Do-Sport(x) $\leftarrow$ Chest-Pain(x),
Go-to-doctor(x) $\leftarrow$ Disease(x, 'Heart-disease') $\wedge$ Do-Sport(x),
$\leftarrow$ Disease(x, 'Heart-disease') $\wedge$ Untroubled(x),
$\leftarrow$ Disease(x, 'Lung-disease') $\wedge$ Do-Sport(x) }

The intended meaning of this example is the following: The base predicate informs us about the people whose chests hurt. The first rule says that if a person has a pain in his chest it may be because he has either a heart disease or a lung disease. The second and fourth rules say that if the chest of a person hurts, then he always practices a sport and, perhaps, goes to the doctor. The information that a person is untroubled if he goes to the doctor is represented by the third rule. The fifth rule says that a person goes to the doctor if he has a pain in his heart and does sport. Finally, the sixth and seventh rules, which are constraints, state that a person cannot have a heart disease and be untroubled, and that a person cannot have a lung disease and do sport, respectively.

In this simple example, it is not difficult to detect that the scheme is trivially consistent, even without using mechanical tools. However, we think that this undesirable characteristic would be very hard to detect if the scheme were more complex and extensive than the previous one. Moreover, schemes which do not satisfy some of the properties, may be easily specified, particularly if there is a large amount of unstructured knowledge represented in them.

Below, we show how our proposal for verifying the consistency of a scheme deals with this example. We use capital letters for identifying both predicates and constants. The unique variable of the scheme is *x*. First, we apply the generation phase, obtaining the following f.o. theory T:

$$
\begin{aligned}
T = \; & \{\text{C-P}(sk_x)\} \\
& \cup \\
& \{\ \text{D}(x,\text{'H'}) \vee \text{D}(x,\text{'L'}) \leftarrow \text{C-P}(x),\ \text{G}(x) \vee \text{D-S}(s) \leftarrow \text{C-P}(x), \\
& \ \ \text{U}(x) \leftarrow \text{G}(x),\ \text{D-S}(x) \leftarrow \text{C-P}(x),\ \text{G}(x) \leftarrow \text{D}(x,\text{'H'}) \wedge \text{D-S}(x), \\
& \ \ \leftarrow \text{D}(x,\text{'H'}) \wedge \text{U}(x),\ \leftarrow \text{D}(x,\text{'L'}) \wedge \text{D-S}(x)\ \}
\end{aligned}
$$

Now, we apply the focusing-on-denials phase. This phase produces a first order theory, f-on-d(T), in which the application of SLT-POS is more efficient for checking whether this theory is (in)consistent:

$$
\begin{aligned}
\text{f-on-d(T)} = \{ & \text{C-P}(sk_x), && 1 \\
& \text{D}(x,\text{'H'}) \vee \text{D}(x,\text{'L'}) \leftarrow \text{C-P}(x), && 2 \\
& \text{G}(x) \vee \text{D-S}(s) \leftarrow \text{C-P}(x), && 3 \\
& \text{U}(x) \leftarrow \text{G}(x), && 4 \\
& \text{D-S}(x) \leftarrow \text{C-P}(x), && 5 \\
& \text{G}(x) \leftarrow \text{D}(x,\text{'H'}) \wedge \text{D-S}(x), && 6 \\
& \text{inconsistent} \leftarrow \text{D}(x,\text{'H'}) \wedge \text{U}(x), && 7 \\
& \text{inconsistent} \leftarrow \text{D}(x,\text{'L'}) \wedge \text{D-S}(x)\ \} && 8
\end{aligned}
$$

Below (Fig. 3), there is an SLT-POS refutation from ←inconsistent in f-on-d(T) (the numbers on the left are used for indicating which clause is used as input clause in each step).

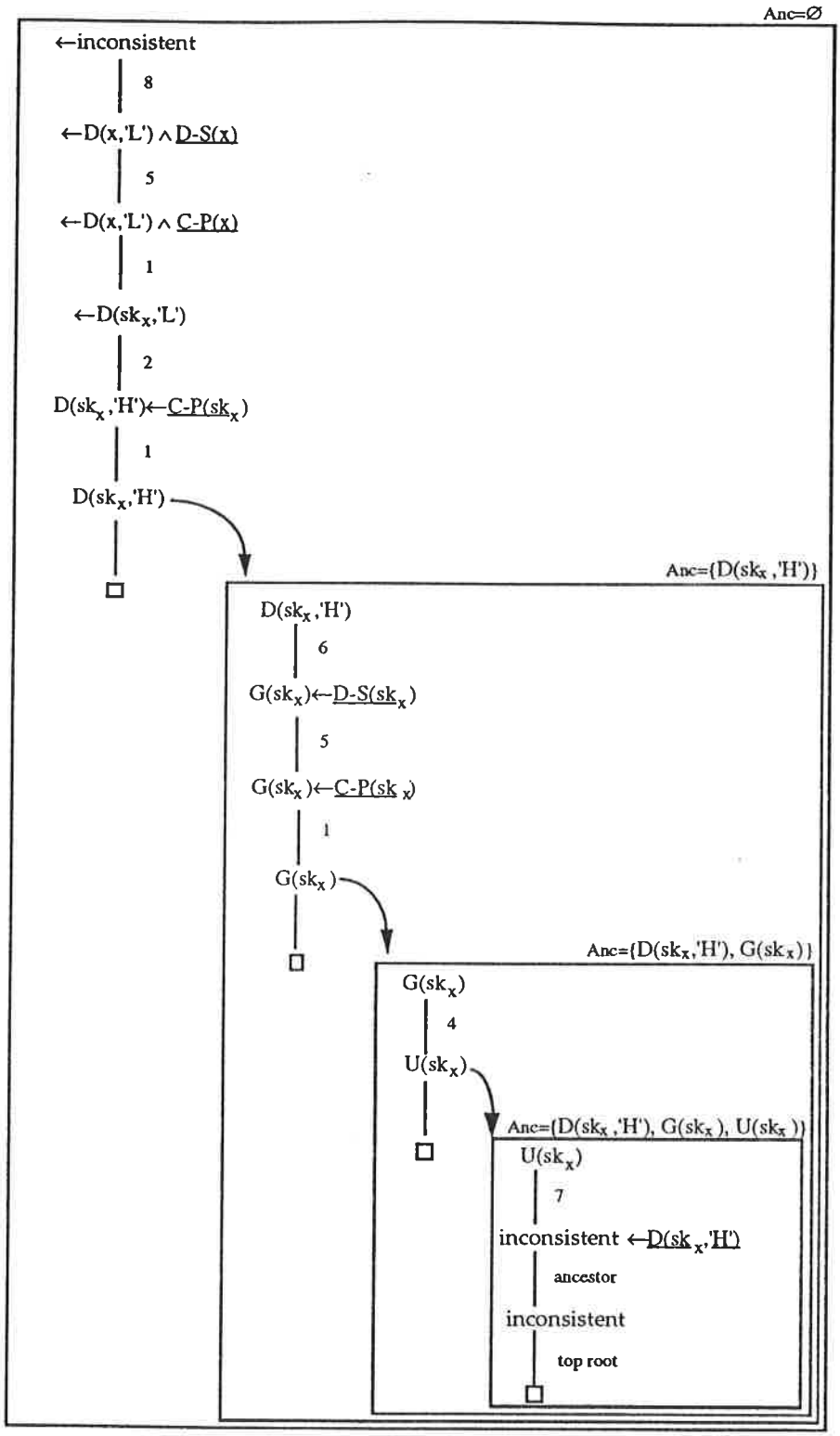

Fig. 3: SLT-POS refutation from ←inconsistent in f-on-d(T))

Since there exists an SLT-POS refutation from ←inconsistent in f-on-d(T), then T is inconsistent, by theorem 4 a). Hence, the scheme MD is either inconsistent or trivially consistent.

On the other hand, the figure below (Fig 4) shows that there is a finitely failed SLT-POS tree from ←inconsistent in f-on-d(Cl). The f. o. theory f-on-d(Cl) is also shown below.

f-on-d(Cl)= { $D(x,'H') \vee D(x,'L') \leftarrow C\text{-}P(x)$, 1

$G(x) \vee D\text{-}S(s) \leftarrow C\text{-}P(x)$, 2

$U(x) \leftarrow G(x)$, 3

$D\text{-}S(x) \leftarrow C\text{-}P(x)$, 4

$G(x) \leftarrow D(x,'H') \wedge D\text{-}S(x)$, 5

$inconsistent \leftarrow D(x,'H') \wedge U(x)$, 6

$inconsistent \leftarrow D(x,'L') \wedge D\text{-}S(x)$ } 7

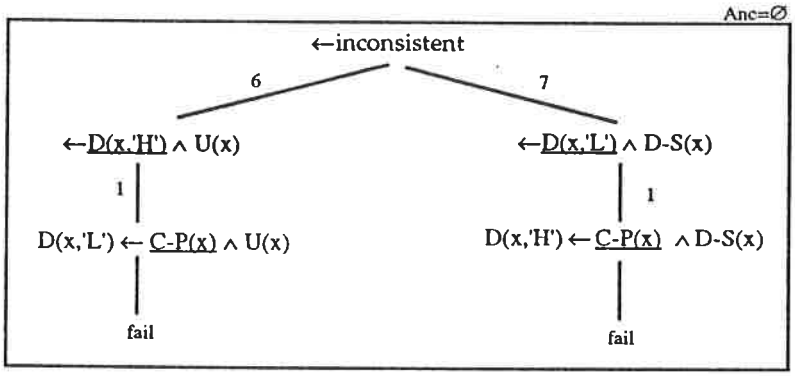

Fig. 4: finitely failed SLT-POS tree from ←inconsistent in f-on-d(Cl)

Therefore, by corollary 1 b), the scheme MD is consistent. From these two previous results, we conclude that MD is a trivially consistent scheme. Moreover, since there is a unique base predicate in our example, MD is also strongly consistent.

CONCLUSIONS

The representation for deductive databases that we have chosen is based on first order logic, generalizing the classical concept of a deductive (definite) database to the concept of a first order (disjunctive) database. We think of a DDB as an instance of some scheme that represents a division between the base or ground knowledge and the deductive knowledge (possibly containing integrity constraints).

One of the aims of this paper has been the definition of some scheme properties, such that their accomplishments by a given scheme help to ensure the correctness of the intended meaning of the scheme. The properties we have defined throughout this paper are consistency, non-trivial consistency, strong consistency, and P-consistency. Consistency is like satisfiability in first-order Logic. Therefore, a consistent scheme guarantees the existence of at least one consistent DDB, instance of this scheme. Non-trivial consistency ensures the existence of a consistent DDB with

ground instances of some base predicate. Strong consistency ensures the existence of a consistent DDB with ground instances of every base predicate scheme. Finally, given a subset P of the base predicate schemes, P-consistency guarantees the existence of a consistent DDB with ground instances of each of the base predicate schemes in P.

Also, we have proposed a method for verifying the accomplishment of the different properties by a given scheme. Essentially, this method consists of generating a f.o. theory from the given scheme, in accordance with the property to be verified, and studying the (in)consistency of this f.o. theory. The centerpiece of this (in)consistency study is a resolution-based theorem prover called SLT-POS resolution, which is an instance of a paradigm for automated (monotonic) reasoning called SL* resolution. Since the deductive knowledge represented in the DDBS is usually specified by near-Horn theories (i.e. theories that slightly differ from Horn theories), we have proposed using SLT-POS resolution because it is better suited for reasoning in these theories. Other instances of the SL* paradigm (like SLT-ALL, also presented in the paper) could be more adequate for reasoning in other classes of theories.

A possible extension of our work is to apply our method to verify the consistency of first order databases. As we said before, there might be inconsistent databases, instances of schemes, which accomplish some, or even all, the properties defined. Therefore, this consistency verification will be needed in the manipulation of the databases. Clearly, for improving the efficiency of this verification, it should be led by the process of knowledge assimilation.

A pending problem of our work is how general constraints can be expressed in the formalism used. That is, suppose W is a general constraint expressed by a closed well-formed-formula. How can W be transformed to be incorporated to our formalism in a sound manner?

ACKNOWLEDGMENT

We thank Hendrik Decker for a lot of valuable comments.

REFERENCES

[Ca] J.C. Casamayor, "Nuevos paradigmas de razonamiento basado en resolución en teorías computacionales", *draft*, DSIC, Univ. Politéc. Valencia (Spain), 1993. To be submitted as Ph.D thesis.

[CDM] J.C. Casamayor, H. Decker and F. Marqués, "A Mechanism for Verification of Knowledge Base Scheme Specifications", in *Proc. European Sym. on the Validation and Verification of Knowledge Based Systems*, Palma de Mallorca (Spain), pp. 103-115, March 1993.

[CL] C.L. Chan and R.C.T. Lee, "Symbolic Logic and Mechanical Theorem Proving", Academic Press, 1973.

[De] H. Decker, "Foundations of First-Order Databases", Siemens ZFE BT SE 24, 1992.

[DC] H. Decker and J.C. Casamayor, "A Prolog-like paradigm for reasoning in first-order theories", in GULP'93 *8th National Conference on Logic Programming*. Gizzeria (Italy), pp. 217-233. June 1993.

[GJ] M.R. Garey and D.S. Johnson, "Computers and Intractability. A Guide to the Theory of NP-Completeness", W.H. Freeman and Company, 1979.

[Ko1] R.A. Kowalski, "Predicate logic as a programming language", in *Proc. IFIP' 74* pp. 569-574, North Holland, 1974.

[Ko2] R.A. Kowalski, *Logic for Problem Solving*, North Holland, 1979.

[KK] R. Kowalski and D. Kuehner, "Linear resolution with selection function", *J. Artificial Intelligence* 2, 227-260, 1971.

[LMR] J. Lobo, J. Minker and A. Rajasekar, *Foundations of Disjunctive Logic Programming*, MIT Press, 1992.

[Ll] J.W. Lloyd, *Foundations of Logic Programming*, Springer, 1987.

[Lo1] D.W. Loveland, "Automated Theorem Proving: a Logical Basis", North Holland 1978.

[Lo2] D.W. Loveland, "Near-Horn Prolog and Beyond", in *J. of Automated Reasoning* 7:1-26, 1991.

[Pl] D.A. Plaisted, "A Sequent-Style Model Elimination Strategy and Positive Refinement, in *J. Automated Reasoning* 6:389-402, 1990.

[Ro] J.A. Robinson, "A machine-oriented logic based on the resolution principle", in *J. ACM* 12:23-41, 1965.

[St1] M.E. Stickel, "A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler", in *J. Automated Reasoning* 4:353-380, 1988.

[St2] M.E. Stickel, "A Prolog technology theorem prover: a new exposition and implementation in Prolog", in *Theoretical Computer Science* 104:109-128, 1992.

[Ul] Ullman, J.D., *Principles of Database and Knowledge Systems*, Computer Science Press, Rockville, Maryland, 1988.