A complete proof procedure for efficient integrity checking in deductive databases

Georg Nüssel[†], Hendrik Decker[‡], Matilde Celma[*] and Juan Carlos Casamayor[*]

[†] Mathemat. Institut, Ludwig-Maximilians-Univ. München, Theresienstr. 39, D-8000 München 2

[‡] Siemens AG, ZFE BT SE 24, P.O. box 83 09 53, D-8000 München 83, hendrik@ztivax.zfe.siemens.de

[*] DSIC, Universidad Politécnica, P.O. box 22 012, E-46020 Valencia, mati@dsic.upv.es

Abstract

We present a proof procedure for checking the integrity of a deductive database. We follow the proposal of Sadri and Kowalski. Our procedure replaces the latter's meta-level rules by inference rules that are implemented more easily on the resolution level. Also, we adapt and extend conditions that are known to ensure completeness and termination, and show that our procedure is complete and terminates for checking violation and satisfaction of integrity in large classes of databases.

Introduction

In /SK/ /KSS/, a theorem proving approach to integrity checking in deductive databases /GMN/ /Ll/ is proposed. The approach, in short: It is assumed that the current state satisfies integrity. By rooting the reasoning process at clauses from a given update, integrity checking is focused on those facts, rules and constraints that are actually affected by the update. So, a considerable simplification is achieved. Generally, this has been established as a principle in /Ni/.

Several other approaches, all based on /Ni/, also simplify integrity checking by focusing on affected data, e.g. /De1/ /LST/ /CCD/. In any of these approaches, phases of *generation* of simplified instances of integrity constraints and phases of *evaluation* of the latter, can be distinguished. An advantage of /SK/ over other methods is that no meta programming is necessary for accomplishing the generation of simplified instances. Rather, both generation and evaluation steps are taken on the resolution level of the proof procedure. For processing deletions, a significant amount of meta programming is encapsulated by meta-inference rules of the theorem prover in /SK/.

In section 2, we present a Selection-driven Linear resolution procedure for Integrity Checking in deductive databases, called SLIC resolution. It is conceived along the lines of /SK/, but some of the latter's meta-level rules are implemented by inference steps that are performed directly on the resolution level.

There are several known results about the soundness and the completeness of various methods for integrity checking /LST/ /SK/ /CCD/. As in /CCD/, soundness and completeness of both generation and evaluation may be distinguished. Also in /Kü/, the generation phase and its completeness is tackled as a separate issue. More appropriate for bringing out the contributions in section 3 of this paper, however, is a distinction between completeness of checking *integrity violation*, on one side, and *integrity satisfaction*, on the other (while soundness is less of a problem, altogether). Actually, this distinction makes sense for all approaches to integrity checking, but usually is glossed over, with the exceptions of /Nü/ /Ce/ /CCD/.

While proof procedures may be complete for finding refutations (that indicate violation in /SK/ /CCD/, and satisfaction in /De1/ /LST/), they may not necessarily terminate with failure to do so (which would indicate the complementary property to violation and satisfaction, respectively). Completeness and termination results for SLDNF resolution /Ll/ are reviewed and adapted to SLIC resolution, in section 4.

1 Preliminaries

An *integrity theory* is a finite set of integrity constraints. Each integrity constraint is supposed to be represented as a *denial* (a clause with empty head). Using transformations of arbitrary formulae into clause form as described in /LT/ /De3/, such a representation is always possible. Generally, a *clause* is written in the form of $A \leftarrow B$ where the head A (if not empty) is a positive literal, the body B is a conjunction of n (possibly negative) literals ($n \geq 0$), and each variable in the clause is supposed to be universally quantified at the front. Clauses may contain function symbols. A *database clause* is a clause with non-empty head. A (*deductive*) *database* is a finite set of database clauses. For a database D, an integrity theory IC is *satisfied in* D if $comp(D) \cup IC$ is a consistent set of formulae, where $comp(D)$ is the well-known *completion* of D /Cl/ /Ll/. Conversely, IC is *violated in* D if $comp(D) \cup IC$ is inconsistent. An *extended clause* is of form $L \leftarrow B$, where the head L (if not empty) is a (possibly negative) literal and the body B is as above. If the head of an extended clause C is positive or absent, then C is also called an *insert clause*; if the head of C is negative, then C is also called a *delete clause*. The definition of extended clauses is due to /SK/.

An *update* (sometimes called *transaction*, in the literature) is a bipartite set $U = U_{del} \cup U_{ins}$ of clauses such that no variant of any clause in U_{del} is in U_{ins} and vice-versa. For a database D, an integrity theory IC and an update U, the *updated database* D' and the *updated integrity theory* IC' are obtained by deleting each database clause in U_{del} from D, deleting each denial in U_{del} from IC, inserting each database clause in U_{ins} to D and inserting each denial in U_{ins} to IC, by an indivisible operation. For convenience, we generally assume that IC' is a mutually consistent set of formulae (see also section 4 of /De5/).

Note that, for a clause C in U_{del}, it may well happen that some consequence of C is still derivable in the updated database, via other clauses. Therefore, we define, for a database D and an update $U = U_{del} \cup U_{ins}$:

> The *actual update* U^* contains each clause in U_{ins}. For each database clause of form $A \leftarrow B$ in U_{del}, U^* contains each delete clause $(\neg A \leftarrow \neg A)\theta$ such that θ is a correct ground answer of *comp*(D) $\cup \{\leftarrow B\}$. U^* contains nothing else.

The condition $\neg A\theta$ appears in the body of delete clauses in U^* obtained from clauses of form $A \leftarrow B$ in U_{del} because our procedure infers that $A\theta$ is deleted if each attempt to prove $\neg A\theta$ in the updated database fails, as we shall see below.

2 SLIC resolution

First, we define SLIC refutations, then finitely failed SLIC trees. We shall see more precisely, later-on, that a SLIC refutation in an updated database, rooted at either an extended clause from the actual update or an integrity constraint, indicates violation of integrity. Conversely, the finite failure of all attempts to refute any of the elements of an actual update or any constraint implies satisfaction.

Definition

Let D be a database, IC an integrity theory, U an update, D' the updated database, IC' the updated integrity theory and C an extended clause in $U^* \cup$ IC'. Then, a *SLIC refutation of* C *in* (D, D', IC') consists of a sequence $C_0 = C, ..., C_n = [\,]$ $(n \geq 0)$ of extended clauses and a sequence $\theta_1, ..., \theta_n$ of substitutions such that, for each $i < n$, a literal L is selected in C_i, and one of the three points below holds.

- L is either selected in the head of C_i or L is positive, and there is a clause C' in D' $\cup$ IC' such that C_{i+1} is a resolvent of C_i and a fresh variant C'' of C' on L and some literal L' in C'', using θ_{i+1} which is an mgu of L and L'.

- L is selected in the body of C_i, L is negative, L is ground, there is a finitely failed SLDNF tree of $D' \cup \{\leftarrow A\}$ where A is the atom of L, C_{i+1} is the resolvent of C_i and L on L, and θ_{i+1} is the identity susbstitution.
- L is selected in the head of $C_i = L \leftarrow B$, and there is a clause of form $A' \leftarrow B'$ in D' with a literal L' in B' such that the polarities of L and L' are opposed, and $C_{i+1} = (\neg A' \leftarrow \neg A' \,\&\, B)\theta_{i+1}$, where $\theta_{i+1} = \theta\phi$, θ is an mgu of the atoms of L and a fresh variant of L', and ϕ is an SLDNF-computed answer of $D \cup \{\leftarrow B'\theta\}$.

Clearly, the first point above corresponds to an ordinary resolution step; L may be selected in the head or in the body of C_i; then, L' occurs in the body or, resp., in the head of C', with polarities of L and L' the same. The second point corresponds to the usual negation-as-failure step in SLDNF resolution. Since the third point never applies if C_i is a denial, it is easy to see that, for a denial C in IC', a SLIC refutation of C in (D, D', IC') coincides with an SLDNF refutation of $D' \cup \{C\}$.

The third point is, in a sense, most interesting: As in the first point, selecting the literal in the head means to propagate a clause through the bodies of clauses via which consequences of the update can be derived. Other than in the first point, the polarities of the two literals on which the inference step are taken are opposed. Propagating an insert (resp., delete) clause through an opposed literal yields a delete (resp., insert) clause. The literal $\neg A'\theta_{i+1}$ in the head of C_{i+1}, which indicates a possible deletion of $A'\theta_{i+1}$, also appears in the body of C_{i+1}, in order to support that the effective deletion of $A'\theta_{i+1}$ is derived if $A'\theta_{i+1}$ cannot be proved. In /SK/, such tests of (non-)provability are done in separate meta-level steps.

In the third point, the answer ϕ is computed in the "old" state D of the database. This ensures that integrity checking is focused on deleted consequences of $A' \leftarrow B'$ that are derivable before the update. Moreover, for each variable x in $\neg A'$, x is grounded in C_{i+1} by ϕ if there is a literal in $B'\theta$ the evaluation of which yields a range of values for x. Note that B may not provide such a literal. For example, if $C_i = (p \leftarrow B)$ and $(A' \leftarrow B') = (q(x) \leftarrow r(x) \,\&\, \neg p)$, then there is no range for x in B.

Also note that C_{i+1} in the third point is always a tautology. Hence, it is easy to see that each step in a SLIC refutation corresponds to a logically correct inference. However, note that each negative literal selected in the body of a clause is processed by the negation-as-failure rule, according to the second point above. Otherwise, negation in the head of clauses is treated as "classical".

Generally, selecting the literal in the head of some extended clause C in a SLIC refutation and deriving a successor from that, is a step in *forward* direction. As opposed to that, selection in the body yields the usual *backward*-directed steps. Forward and backward steps in SLIC resolution correspond to what is done in the generation and, resp., the evaluation phase of other integrity checking methods.

Definition

Let D be a database, IC an integrity theory, U an update, D' the updated database, IC' the updated integrity theory and C_0 an extended clause in $U^* \cup IC'$. Then, a finitely failed *SLIC tree of* C_0 *in* (D, D', IC') is a finite tree such that each of its nodes is a non-empty extended clause, its root is C_0 and the following holds.

- Let C be a non-leaf node in the tree and suppose a literal L is selected in C. Then, each clause derived in one SLIC step from C on L, as defined in the definition of SLIC refutations, modulo variants, is a child node of C.
- Let C be a leaf node in the tree and suppose a literal L is selected in C. Then, none of the three points in the definition of SLIC refutations applies, i.e. no SLIC step can be taken from C. Moreover, if L is negative, ground and selected in the body of C, then there is an SLDNF refutation of $D' \cup \{\leftarrow A\}$, where A is the atom of L.

For brevity, we skip definitions of (not necessarily successful) SLIC derivations and (not necesarily finitely failed) SLIC trees, that describe the search space of SLIC resolution. They can be easily obtained along the lines of definitions for SLDNF /Ll/.

3 Characterizing soundness and completeness of integrity checking

For methods of integrity checking in which phases of generation and evaluation are distinguished, results about their soundness and completeness may be divided up into respective results for each phase. Intuitively, generation is sound if its outcome correctly identifies the data affected by a given update; it is complete if it identifies each integrity constraint that is actually violated by the update. Evaluation is sound if its outcome correctly identifies violation or satisfaction of integrity; it is complete if violation and satisfaction is detected always.

Roughly, SLIC resolution is sound for generation and evaluation since, as shown in /Nü/, each inference step in forward and backward direction produces a logical consequence of the union of database and constraints. In the remainder, we focus on the completeness of SLIC resolution. Distinguishing further-on between completeness of generation and evaluation would not make much sense, since forward and backward steps are interleaved non-deterministically in SLIC resolution.

Rather, for a database D, an integrity theory IC that is satisfied in D, an update U, updated database D' and updated integrity theory IC', we distinguish between completeness of checking *violation* and *satisfaction* of integrity. That may be characterized by the following conditions.

SLIC resolution is complete for checking violation in the standard way if the following holds: If IC' is violated in D', then there is a SLIC refutation of some I∈IC' in (D, D', IC').

SLIC resolution is complete for checking satisfaction in the standard way if the following holds: If IC' is satisfied in D', then, for each I∈IC', there is a finitely failed SLIC tree of I in (D, D', IC').

The characterizations above reflect the standard way of checking integrity, in the sense that the generation phase is skipped and each integrity constraint is evaluated as it stands. It follows from what we have seen before that SLIC and SLDNF resolution coincide for checking integrity the standard way. However, following /SK/, it is, in general, much more efficient to root integrity checking at the clauses of an actual update, rather than at the (unsimplified) constraints. Thus, also the following conditions may be used to characterize completeness.

SLIC resolution is complete for checking violation if the following holds: If IC' is violated in D', then there is a SLIC refutation of some C∈U* in (D, D', IC').

SLIC resolution is complete for checking satisfaction if the following holds: If IC' is satisfied in D', then, for each C∈U*, there is a finitely failed SLIC tree of C in (D, D', IC').

4 Properties for ensuring completeness and termination of SLIC resolution

In general, both SLDNF and SLIC resolution are neither refutation-complete nor do they always terminate. Hence, sufficiently large and practically relevant classes of databases and integrity theories that ensure completenesss and termination of proof procedures are of interest. We are going to redefine some known properties that ensure completeness and termination of SLDNF resolution, below, and then give results of completeness and termination for SLIC resolution.

For transposing completeness results for SLDNF to SLIC, the following proposition is instrumental. It can be shown along the lines of the proof of a similar result in /Nü/. For stating the proposition, we first adapt a well-known definition /Ll/.

Definition Let P be a set of clauses (that may contain denials). P is *allowed* if, for each clause C in P, each variable in C also occurs in a positive literal of the body of C.

Proposition

Let D be a database, IC an integrity theory that is satisfied in D, U an update, D' the updated database and IC' the updated integrity theory, such that each of D, D', IC, IC' is allowed. Further, suppose that, for each $I \in IC$, there is a finitely failed SLDNF tree of $D \cup \{I\}$ and, for some $I' \in IC'$, there is an SLDNF refutation of $D' \cup \{I'\}$. Then, there is an extended clause $C_0 \in U^*$ and a SLIC refutation of C_0 in (D, D', IC').

Next, we are going to review (and slightly generalize) definitions of well-known structural properties of databases such as "definite", "hierarchical", "stratified", etc., adapting them to the context of this paper. As we shall see more precisely, later-on, appropriate combinations of such properties may ensure completeness and/or termination of SLIC (and SLDNF) resolution. As a basis, we use the following definitions of dependencies between clauses (slightly extending similar definitions in /ABW/ /DC/ /De4/ and elsewhere).

Definition Let P be a (possibly infinite) set of clauses and C, C' a pair of clauses in P.

a) C *depends on* C' *in one step* if there is an atom in some literal in the body of C that unifies with the head of a fresh variant of C'.

b) C *depends on* C' if either C depends on C' in one step, or C depends on some clause C'' in P in one step and C'' depends on C'.

c) C *depends positively* (resp., *negatively*) on C' if C depends on C' and that dependency does not go through any negative literal (resp., goes through at least one negative literal).

d) C *depends evenly* (resp., *oddly*) on C' if C depends on C' and that dependency goes through an even (resp., odd) number of negative literals.

Note that, by the definition above, each clause that depends oddly on itself also depends evenly on itself (by running the cycle twice). For convenience (and unlike some similar definitions in the literature), dependencies are not defined such that each clause would necessarily depend on itself.

Definition Let P be a set of clauses and $\underline{P}$ the set of ground instances of clauses in P.

a) P is *definite* if no clause in P contains a negative literal in its body. (A somewhat more general definition of P *definite* in /DC/ requires that no clause in P depends negatively on any clause in P).

b) P is *hierarchical* if no clause in P depends on itself.

c) P is *locally hierarchical* if there is a mapping |.| assigning each clause in $\underline{P}$ a countable ordinal such that the following holds: For each pair C, C' of clauses in $\underline{P}$ such that C depends on C', $|C| > |C'|$.

d) P is *acyclic* if there is a mapping |.| assigning each clause in $\underline{P}$ a natural number such that the following holds: For each pair C, C' of clauses in $\underline{P}$ such that C depends on C', $|C| > |C'|$.

e) P is *stratified* if no clause in P depends negatively on itself.

f) P is *locally stratified* if there is a mapping |.| assigning each clause in $\underline{P}$ a countable ordinal such that, for each pair C, C' of clauses in $\underline{P}$, the following holds: If C depends on C', then $|C| \geqq |C'|$, and if C depends negatively on C', then $|C| > |C'|$.

g) P is *call-consistent* if no clause in P depends oddly on itself.

h) P is *locally call-consistent* if there is a mapping |.| assigning each clause in $\underline{P}$ a countable ordinal such that, for each pair C, C' of clauses in $\underline{P}$, the following holds: If C depends on C', then $|C| \geqq |C'|$, and if C depends both evenly and oddly on C', then $|C| > |C'|$.

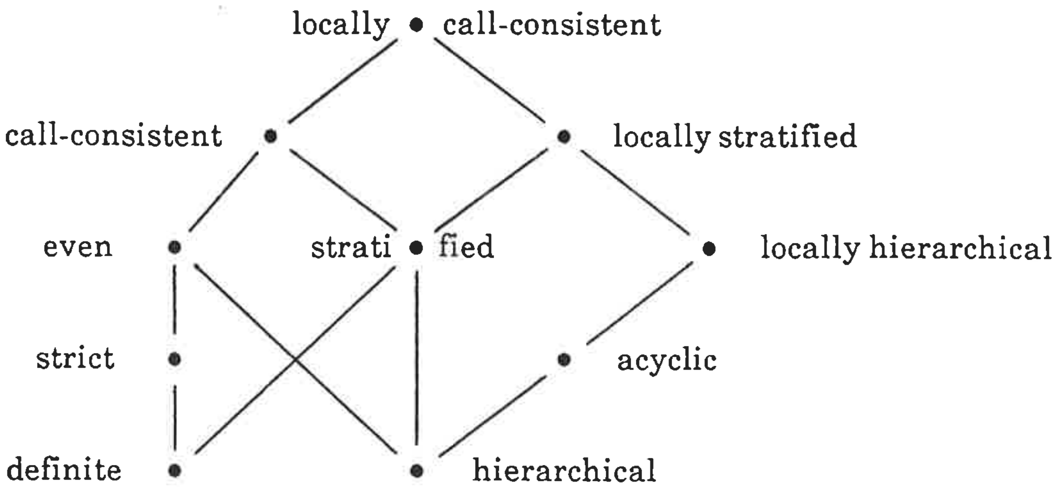
i) P is *strict* if there is no pair C, C' of clauses in P such that C depends both evenly and oddly on C'.

j) P is *even* if there is no pair C, C' of clauses in P such that C depends both evenly and oddly on C' and C' depends on itself.

Similar definitions of the properties above can be found in the following references: *a), b)* in /Cl/ /Ll/; *c), d), h)* in /Ca/; *e), i)* in /ABW/; *f)* in /Pr/; *g)* in /Sa/; *j)* in /DC/; definitions similar to *e), g), i), j)* are also in /Ku/, and *h)* has been defined in /Sa/ (under the name of *order-consistent*).

Although there are, to our knowledge, no completeness (resp., termination) results for SLDNF resolution in locally stratified or locally call-consistent databases that would generalize known results for more restricted classes, we have included definitions *f)* and *h)* above to round off the picture. Moreover, the consistency of the completion *comp*(D) of a locally call-consistent database D has been shown in /Sa/. Consistency of *comp*(D) clearly is an interesting property, and is going to be required for the main results of the next section.

If, in the diagram below, there is an edge from property X to property Y in upward direction, then Y generalizes X.



For both completeness and termination, the propery of "boundedness" of denials /AB/ /Ca/ is of interest, as well as a property called "weakly bounded" /Nü/. We adapt both properties, as follows.

Definition Let P be a set of clauses and $\underline{P}$ the set of ground instances of clauses in P.

a) A denial C is *bounded in* P if $\underline{P}$(C) ∪ {C} is acyclic, where $\underline{P}$(C) is the set of clauses in $\underline{P}$ on which C depends. (The usual, slightly more restrictive definition of boundedness essentially requires that P ∪ {C} is acyclic).

b) P is *weakly bounded* if, for each clause C in P and each variable x in C, there is a positive literal L in the body of C such that x occurs in L and ← L is bounded in P.

Although the definitions above are slight generalizations of namesakes in the literature, it can be shown that the essential entailments of the respective properties are preserved. In particular, that holds for the completeness and termination results of SLDNF resolution, referred to in the theorem about completeness and termination of SLIC resolution, below. For stating that theorem, we also need to adapt the concept of *floundering* /Ll/, as follows.

Definition Let D be a database and Q a query. D ∪ {Q} is *straight* (sometimes called *flounder-free*) if no SLDNF derivation of D ∪ {Q} would ever reach a non-empty clause consisting of non-ground negative literals only. D is *straight* if, for each atom A, D ∪ {← A} is straight. For an integrity theory IC, D ∪ IC is *straight* if, for each I in IC, D ∪ {I} is straight.

5 Completeness of integrity checking with SLIC resolution

With the definitions of section 4, we are in the position to state the following results about the completeness of integrity checking with SLIC resolution. They follow from the proposition in section 4 and corresponding results of completeness and, resp., termination of SLDNF resolution, as referenced behind the points below.

Theorem

Let D be a database such that *comp*(D) is consistent, IC a consistent integrity theory such that, for each I ∈ IC, there is a finitely failed SLDNF tree of D ∪ {I} (hence IC is satisfied in D), U an update, D' the updated database and IC' the updated integrity theory. Further, for I ∈ IC, let D'(I) denote the set of clauses in D' on which I depends. Then the following holds.

a) SLIC resolution is complete for checking violation if, for each I ∈ IC', D'(I) fulfills each of the required properties at any one of the following points.

- definite /Ll/
- hierarchical, straight /Sh/
- stratified, strict, allowed /CL/
- even, allowed /Ku/ /DC/
- acyclic, straight /Ca/

b) SLIC resolution is complete for checking satisfaction if, for each I ∈ IC', D'(I) fulfills each of the required properties at any one of the following points.

- hierarchical, straight /Sh/
- acyclic, straight, weakly bounded /Nü/

We remark that the main completeness result in /Ku/ is stated for call-consistent (rather than even) databases, but it additionally requires the strictness of the set of clauses on which given queries (constraints) depend.

The completeness result in /SK/ corresponds to the first point in part *a)*, above. Clearly, the theorem entends this result to many other interesting classes of databases, and also deals with the completeness of checking satisfaction. Acyclicity appears to be the most supportive of properties, for the completeness of checking violation and satisfaction of integrity. As argued in /Ca/, acyclicity allows much of the recursion that arises in practice, such as the commonly used inductive definitions of the append, member and reverse predicates. Also note that, in part *b*), the second point properly generalizes the first one.

With regard to the last point of the theorem above, /Nü/ shows that, for a finite, weakly bounded set of clauses, finitely failed SLIC trees can be obtained by selecting literals in SLIC derivations in a "fair" manner For a SLIC derivation d, selection of literals is *fair* if either d is failed or, for each literal L in d, some instance of L is selected after a finite number of steps. Note that this fairness condition essentially is the simple one of /LI/ (and not the unpractical one of /CL/).

There are two other termination results that are of interest if integrity checking is rooted at (possibly simplified) integrity constraints. Firstly: For a locally hierarchical database D and a denial (constraint) C that is bounded in D, it follows from /AB/ /Ca/ that each SLDNF tree, and hence each SLIC tree rooted at C, is finite.

Secondly, /Nü/ defines: For a locally hierarchical database D, a denial C is *weakly bounded in* D if, for each variable x in C, there is a positive literal L in C such that x occurs in L and $\leftarrow$ L is bounded in D. /Nü/ shows that, for an acyclic database D and a denial C that is weakly bounded in D, each fair SLDNF tree and each fair SLIC tree rooted at C is finite.

As for straightness and (weak) boundedness, it is interesting to note that both properties are not decidable, in general. Also, acyclicity is not, while all other properties required in any of the points in the theorem above are. It is shown in /LI/ and others that allowedness entails straightness. In /LI/, another decidable allowedness condition that is weaker than the definition of allowed, above, is specified, and shown to entail straightness. In /De4/, two relaxations of the weak allowedness condition of /LI/, called *positively covered* and *straight-covered*, are introduced. Each of these generalizations is decidable, entails straightness and can be easily adapted to the framework of this paper. For brevity, we do not do that explicitly, here, but leave it with the reference.

Note that the allowedness condition in the third and fourth point of part *a*), above, cannot be replaced by straightness, nor by any of the weakenings of allowedness mentioned so far. For example, if $D = \{p(x) \leftarrow \neg q(x)\}$, $I = \leftarrow \neg p(f(a))$

and $U = U_{ins} = q(f(x))$, then both $D \cup \{I\}$ and $D' \cup \{I\}$ are positively covered. Since they are also hierarchical, it follows that, for both states, integrity checking in the standard way is complete and terminates properly. However, integrity checking with SLIC resolution flounders if the process is rooted at the update, since the computation of answers of $D \cup \{\leftarrow \neg q(x)\}$ is required for obtaining successor clauses of $q(f(x)) \leftarrow$, when a fresh variant of $p(x) \leftarrow \neg q(x)$ is used as input clause in the derivation of $q(f(x)) \leftarrow$ in $(D, D', \{I\})$, where $D' = D \cup \{q(f(x)) \leftarrow\}$.

Significant generalizations of allowedness that are decidable, entail straightness and preserve refutation-completeness of SLDNF resolution can be found in /DC/ /CD/ /De4/. It should be interesting to investigate whether they can be adapted such that also the completeness of SLIC resolution for violation of integrity is preserved.

Last, we recall that, for an update U, the delete clauses in the actual update U* are obtained by computing ground answers for conditions of delete clauses in U. For a database D and an integrity theory IC, a sufficient condition for the groundness of computed answers in SLDNF - and SLIC resolution is that $D \cup IC$ is (weakly) allowed. Also sufficient for groundness is that $D \cup IC$ is positively covered (while neither that $D \cup IC$ is straight nor that $D \cup IC$ is straight-covered ensures groundness, in general). However, the property of *ground-covered* /De4/ generalizes the weak allowedness condition in /Ll/, and entails both groundness and straightness.

6 An example

We illustrate SLIC resolution with an example of a database and an integrity theory including the clauses below (and possibly others, that are not relevant).

Each fact in the room relation consists of the number of a room (first component) that belong to a certain department (second component). Facts in the course relation consist of subject names (first component) and the organizing department (second). Each fact in the lecture relation contains course subject (first), room number (second), weekday (third) and start time (fourth). The r-equipment relation specifies pieces of equipment (second) in rooms (first). The c-equipment relation specifies pieces of equipment (second) necessary for certain courses (first).

The three rules declare certain rooms to be either inadequate or acceptable for certain courses. Note that (perhaps because of scarcity of space) a room may have to be considered acceptable for some course, even though it might qualify as inadequate for that course. Also note that the maths and the computer science departments may use each other's rooms if acceptable and consistent with the integrity constraints. The first constraint denies that there may be two different lectures in the same room at the same time of the same day. The second one forbids that any lecture be held in a room that is not acceptable for the respective course. It is easy to see that integrity is satisfied in the given state.

All variable symbols are represented by (possibly subscribed) capital letters. All constants are strings of digits or lower case characters. The symbol "←" is omitted in clauses that are facts. An expression of form $E \neq F$, as used in one of the rules below, is an abbreviation for the literal ¬equal(E, F), where the predicate equal is supposed to be defined by the (possibly built-in) clause equal(E, E) ←. For later reference, clauses are numbered.

1 room(27, maths)	4 course(logic, maths)	7 lecture(logic, 27, mon, 10)
2 room(36, maths)	5 course(prolog, comp-sc)	8 lecture(prolog, 36, tue, 10)
3 room(43, comp-sc)	6 course(lisp, comp-sc)	9 lecture(lisp, 43, wed, 11)

10 r-equipment(27, board)	15 c-equipment(logic, board)
11 r-equipment(27, coffee-machine)	16 c-equipment(logic,coffee-machine)
12 r-equipment(36, board)	17 c-equipment(prolog, overhead)
13 r-equipment(36, overhead)	18 c-equipment(prolog, board)
14 r-equipment(43, video)	19 c-equipment(lisp, overhead)

20	inadequate(R, C)	←	room(R, X) & course(C, Y) & c-equipment(C, E) & ¬r-equipment(R, E)
21	acceptable(R, C)	←	room(R, X) & course(C, X)
22	acceptable(R, C)	←	¬inadequate(R, C)
23		←	lecture(C_1, R, W, S) & lecture(C_2, R, W, S) & $C_1 \neq C_2$
24		←	lecture(C, R, W, S) & ¬acceptable(R, C)

U_1 = {lecture(prolog, 27, wed, 11)} U_2 = {¬ r-equipment(43, video)}

U_3 = {¬ r-equipment(36, overhead)}

Note that the database above is not allowed, because of the second rule about the predicate acceptable. However, the union of database and integrity theory is allowed, in the sense of the weak allowedness condition of /LI/, since acceptable never occurs in a positive literal of the body of any rule or constraint.

The update U_1 above leads to an inconsistent state. Proving that by running SLIC resolution coincides with doing the same by the procedure of /SK/. On the other hand, U_2 does not violate integrity. That can also be shown by running SLIC.

The SLIC refutation below shows that U_3 (removing the overhead projector from room number 36) violates integrity. Its root is the only element in U_3*. Used input clauses are identified by their respective numbers. Selected literals are underlined, if there is a choice. Substitutions are understood.

$\neg$ r-equipment(36, overhead) ← <u>$\neg$r-equipment(36, overhead)</u>

| *negation-as-failure*

$\neg$ r-equipment(36, overhead) ←

| 20

inadequate(36, C) ← <u>room(36, X)</u> & course(C, Y) & c-equipment(C, overhead)

| 2

inadequate(36, C) ← course(C, Y) & <u>c-equipment(C, overhead)</u>

| 17

inadequate(36, prolog) ← <u>course(prolog, Y)</u>

| 5

inadequate(36, prolog) ←

* | 22

$\neg$acceptable(36, prolog) ← <u>$\neg$acceptable(36, prolog)</u>

| *negation-as-failure, see subsidiary tree below*

$\neg$acceptable(36, prolog) ←

| 24

← lecture(prolog, 36, W, S)

| 8

[]

```
                    ←  acceptable(36, prolog)

                    21 /                    \ 22

← room(36, X) & course(prolog, X)          ← ¬inadequate(36, prolog)

          | 2                                    | negation-as-failure, see below

   ← course(prolog, maths)                      failure

          |

      failure
```

```
                    ← inadequate(36, prolog)

                              | 20

← room(36, X) & course(prolog, Y) & c-equipment(prolog, E) & ¬r-equipment(36, E)

                              | 2

   ← course(prolog, Y) & c-equipment(prolog, E) & ¬r-equipment(36, E)

                              | 5

         ← c-equipment(prolog, E) & ¬r-equipment(36, E)

                              | 17

         ← ¬r-equipment(36, overhead)

                              | negation-as-failure

                              [ ]
```

Note that, in the derivations above, a literal in the head is selected only if there is no selectable literal in the body. This should be easily implementable on top of any selection function for SLDNF (which only has to deal with literals in the body). Further note that selection in the body of clauses above prioritizes literals that are more instantiated than others, and proceeds from left to right in case of equal priorities. For the restricted case of SLDNF resolution, this is a well-known strategy that has turned out to often be of advantage in deductive databases.

Further note that the step marked *, above, is taken only after a subsidiary refutation of D ∪ {← adequate(36, prolog)} is computed, according to the third step of the definition of SLIC refutations, where D denotes the database before the update.

Concluding remarks

The last word on integrity checking has not been spoken yet. One area that has hardly been explored so far (possibly excepting some beginnings in /De1/ /GL/ /Ol/ /CCD/ and others), is the application of *compilation* and *partial evaluation* /LS/ to the generation of simplified constraint instances from *update schemes*, and the evaluation of such instances. Also, the particular difficulties of integrity checking that arise by the presence of large amounts of data on secondary storage, as discussed in /Wü/, deserve further studies. Possibly, the treatment of updates and constraints may one day be integrated in a proof procedure that handles query answering, updating, integrity maintenance, consistency checking and related features such as exception handling, default reasoning and belief revision in a uniform, but efficient way. For some first steps in this direction, see /KS/ /KKT/ /Cm/.

Acknowledgements

To a large extent, this paper draws from the first author's diploma thesis /Nü/, supervised by Prof. H. Schwichtenberg at LMU München. Independently, ideas of implementing meta-level steps in the procedure of /SK/ more directly on the resolution level had also been sketched in /De2/ at ECRC, München, where the second author benefited a lot from discussions with Rodney Topor and Fariba Sadri. Fariba also gave valuable feedback to an earlier version of this paper. The concepts of soundness and completeness of integrity checking (section 3) have also been developed, independently, in the third author's doctoral thesis /Ce/, supervised by Prof. I. Ramos at DSIC, Valencia.

References

/AB/ K.R. Apt and M. Bezem, Acyclic Programs, in D.H.D. Warren and P. Szeredi (eds), in D.H.D. Warren and P. Szeredi, *Proc. 7th ICLP*, 617-633, MIT Press, 1990.

/ABW/ K.R. Apt, H.A. Blair and A. Walker, Towards a theory of declarative knowledge, in J. Minker (ed), *Foundations of Deductive Databases and Logic Programming*, 89-148, Morgan Kaufman, 1988.

/Ca/ L. Cavedon, Acyclic logic programs and the completeness of SLDNF-resolution, *Theoretical Computer Science* 86, 81-92, 1991.

/CCD/ M. Celma, J.C. Casamayor and H. Decker, Improving integrity checking by compiling derivation paths, DSIC, Univ. Politéc. Valencia, and Siemens AG, ZFE BT SE 24, München, revised version, June 1992.

/CD/ L. Cavedon and H. Decker, A weak allowedness condition that ensures completeness of SLDNF-resolution, in A. Olivé (ed), *Proc. Int'l Workshop on the Deductive Approach to Information Systems and Databases*, 153-171, Report de recerca LSI 90/30, Univ. Politèc. Catalunya, Barcelona (Spain), 1990.

/Ce/ M. Celma, *Comprobacion de la Integridad en Bases de Datos Deductivas*, PhD thesis, DSIC, Universidad Politécnica de Valencia, 1992.

/CL/ L. Cavedon and J.W. Lloyd, A completeness theorem for SLDNF resolution, *J. Logic Programming* 7, 177-191, 1989.

/Cl/ K.J. Clark, Negation as failure, in H. Gallaire and J. Minker (eds), *Logic and Databases*, 293-322, Plenum Press, 1978.

/Cm/ J.C. Casamayor, An abductive proof procedure for hypothetical reasoning in computational theories, DSIC, Universitat Politécnica de Valencia, 1992, also in this volume.

/DC/ H. Decker and L. Cavedon, Generalizing allowedness while retaining completeness of SLDNF-resolution, in E. Börger et al (eds), *Proc. 3rd Workshop on Computer Science Logic*, 98-115, Springer LNCS 440, 1990.

/De1/ H. Decker, Integrity enforcement on deductive databases, in L. Kerschberg (ed), *Expert Database Systems*, 381-395, Benjamin-Cummings, 1987.

/De2/ H. Decker, A note on enforcing integrity with a resolution-like proof procedure, *Working Paper KB-2*, ECRC, 1987.

/De3/ H. Decker, The range form of databases and queries, or: How to avoid floundering, in J. Retti und K. Leidlmair (Hrsg.), *Proc. 5. Österreichische Artificial-Intelligence-Tagung*, Igls (Tirol), 114-123, Springer Informatik-Fachberichte 208, 1989.

/De4/ H. Decker, On generalized cover axioms, in K. Furukawa (ed), *Proc. 8th ICLP*, 693-707, MIT Press, 1991.

/De5/ H. Decker, Knowledge assimilation in deductive databases -- an overview, Siemens AG, ZFE BT SE 24, 1992, also in this volume.

/GL/ U. Griefahn and S. Lüttringhaus, Top-down integrity checking for deductive databases, in D.H.D. Warren and P. Szeredi, *Proc. 7th ICLP*, 130-144, MIT Press, 1990.

/GMN/ H. Gallaire, J. Minker, J.M. Nicolas, Logic and databases: A deductive approach, *Computing Surveys* 16, 153-185, 1984.

/KKS/ A.C. Kakas, R.A. Kowalski and F. Tony, Abductive logic programming, *Draft*, Imperial College, London (England), 1992.

/KS/ R.A. Kowalski and F. Sadri, Logic programs with exceptions, in D.H.D. Warren and P. Szeredi, *Proc. 7th ICLP*, 598-613, MIT Press, 1990.

/KSS/ R.A. Kowalski, F. Sadri and P. Soper, Integrity checking in deductive databases, *Proc. 13th VLDB*, 61-69, 1987.

/Ku/ K. Kunen, Signed data dependencies in logic programs, *J. Logic Programming* 4, 289-308, 1987.

/Kü/ V. Küchenhoff, On the efficient computation of the difference between consecutive database states, in C. Delobel, M. Kifer and Y. Masunaga (eds), *Proc. DOOD '91*, 478-502, Springer LNCS 566, 1991.

/Ll/ J.W. Lloyd, *Foundations of Logic Programming*, 2nd edition, Springer, 1987.

/LS/ J.W. Lloyd and J.C. Shepherdson, Partial evaluation in logic programming, *J. Logic Programming* 11, 217-242, 1991.

/LST/ J.W. Lloyd , E.A: Sonenberg and R.W. Topor, Integrity constraint checking in stratified databases, *J. Logic Programming* 4, 331-343, 1987.

/LT/ J.W. Lloyd and R.W. Topor, Making Prolog more expressive, *J. Logic Programming* 1, 225-240, 1984.

/Ni/ J.M. Nicolas, Logic for improving integrity checking in relational data bases, *Acta Informatica* 18, 227-253, 1982.

/Nü/ G. Nüssel, Integritätstests in deduktiven Datenbanken - ein beweistheoretischer Ansatz, *Diploma Thesis*, Mathematische Fakultät, LMU München, 1991.

/Ol/ A. Olivé, Integrity constraints checking in deductive databases, *Proc. 17th VLDB*, 1991.

/Pr/ T.C. Przymusinski, On the declarative semantics of deductive databases, in J. Minker (ed), *Foundations of Deductive Databases and Logic Programming*, 193-215, Morgan Kaufman, 1988.

/Sa/ T. Sato, Completed logic programs and their consistency, *J. Logic Programming* 9, 33-44, 1990.

/Sh/ J.C. Shepherdson, Negation in logic programming, in J. Minker (ed), *Foundations of Deductive Databases and Logic Programming*, 19-88, Morgan Kaufman, 1988.

/SK/ F. Sadri and R.A. Kowalski, A theorem-proving approach to database integrity, in J. Minker (ed), *Foundations of Deductive Databases and Logic Programming*, 313-362, Morgan Kaufman, 1988.

/Wü/ B. Wüthrich, Large Deductive Databases with Constraints, Informatik-Dissertationen ETH Zürich, Nr. 26, Verlag der Fachvereine, 1991.