9160483

MFDBS 89

2nd Symposium on

Mathematical Fundamentals of Database Systems

华南理工大学 图书馆藏书

53

TP311.13-53
M426
1989

9160483

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

364

J. Demetrovics B. Thalheim (Eds.)

华南理工大学
图书馆藏书

MFDBS 89

2nd Symposium on
Mathematical Fundamentals of Database Systems
Visegrád, Hungary, June 26–30, 1989
Proceedings

E9160483

Springer-Verl

Berlin Heidelb

Editorial Board

D. Barstow W. Brauer P. Brinch Hansen D. Gries D. Luckham
C. Moler A. Pnueli G. Seegmüller J. Stoer N. Wirth

Volume Editors

János Demetrovics
Computer and Automation Institute
Hungarian Academy of Sciences
P.O. Box 63, H-1502 Budapest, Hungary

Bernhard Thalheim
Kuwait University, Department of Mathematics
P.O. Box 5969, 13060 Kuwait

CR Subject Classification (1987): F.4.1, G.2.1, H.2

ISBN 3-540-51251-9 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-51251-9 Springer-Verlag New York Berlin Heidelberg

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in other ways, and storage in data banks. Duplication of this publication or parts thereof is only permitted under the provisions of the German Copyright [illegible]mber 9, 1965, in its [illegible] 1985, and a copyright fee must always be [illegible]man Copyright Law.

Preface

Within the last 20 years the database area has enriched computer science with a set of important practical and deep theoretical results. The main topics in modern database theory are nowadays: theoretical fundamentals of database models (dependency theory, design theory, extensions of the relational, the entity-relationship and semantic models), extended database semantics (complex data objects, databases and logic, extended data semantics and data types), theoretical fundamentals for extended architectures and system support (transaction models, concurrency, data distribution, data access, query, integrity, security).

A great deal of work has been done, and there is a need for intensive exchange of experience. Therefore, the MFDBS symposia are organized every odd year. The first one was held in Dresden in 1987, the second one will be held in Visegrád, Hungary, June 26-30, 1989.

This volume is a collection of the most important contributions of the Visegrád Symposium. The papers selected from more than 100 submissions, originating from 23 countries in 4 continents, can be divided roughly into the following sections:

Theoretical Fundamentals of Relational Databases;
Logical Foundations and Databases;
Data Modelling;
Database Design;
Deductive Databases;
Transaction Management and Security;
Concurrency Control and Distributed Databases.

For their careful evaluation of the submitted papers we should like to express our sincere thanks to the members of the program committee, S. Abiteboul, G. Ausiello, A. Benczúr, J. Biskup, S. Ceri, G. Katona, R. Kirkova, J. Paredaens, V. Pasichnik, A. Riha, Y. Sagiv, J.W. Schmidt, A. Stogniy, M.Y. Vardi, and also to the other reviewers, K. Bähler, F. Bry, H.H. Brüggemann, B. Convent, R. Kemp, U.W. Lipeck, R. Manthey, U. Räsch, H. Stiefeling, and H. Thiele.

Finally we should like to thank Springer-Verlag for the quick publication and the Hungarian National Foundation for Scientific Research, Grant No. 1066 for the partial financial support of the Symposium.

Budapest, Kuwait
March 1989

J. Demetrovics and B. Thalheim
Co-Chairmen

Table of Contents

此为试读，需要完整PDF请访问：www.ertongbook.com

Selective Refutation of Integrity Constraints in Deductive Databases

P. Asirelli, C. Billi, P. Inverardi

Istituto di Elaborazione della Informazione - CNR
via S. Maria n. 46
I-56100 Pisa

1. INTRODUCTION

In the last ten years Logic Programming has been a very active and successful area of research. One effect of that success is that it has made people aware of the advantages of logic, both as a new approach (declarative) to solve old problems and as a new language philosophy to build tools to be integrated with existing applications. One of the interesting area of applications of logic, has been to deductive databases and query languages for databases such as Datalog programs.

This paper falls into the area of deductive databases seen as a Horn clause logic program [Lloyd 87]. With this approach integrity constraints are considered as first order formulas that have to be logical consequences of *suitable* models of the given program. Updates to the database foresee for integrity checking that can be very inefficient.

The aim of the paper is to present an approach to integrity checking that is based on the assumption that, if the database was consistent with respect to the integrity constraints (i.e. its model satisfied the integrity constraint formulas) before the update, then, any possible inconsistency after the update is due to the update itself. One of the approach in the literature is to verify the validity of the Integrity formulas on the new database by SLD-resolution. This can be very inefficient if it has to be done frequently, while such an inefficency could be decreased if, in the SLD-resolution one could consider only resolution steps that are involved with the updates. That is, in terms of SLD-tree, we want to consider only those branches where the inserted/deleted clause appear at some stage, while not considering the others that are assumed to remain unchanged after the update.

We are going to present a deductive strategy to compute, for a given program P, a given goal G and a clause C, the answers to $P \cup \{G\}$ that are obtained by using the clause C. C is said to be the "driving" clause. We call this method "Driven Method" denoted from now on by DM.
The method is based on the SLD-resolution with a new defined Selection Rule.

2. Introduction to the DM method and definitions

2.1 Overview of the DM method

Given a definite program P, i.e. a set of Horn clauses, a definite goal G, i.e. G: $\leftarrow A_1,\ldots,A_n$ where each A_i is an atom, and a clause C in P, the method is a deductive strategy to build a refutation-tree that has G at the root and has the following properties:

- **Soundness**: Every answer is correct: i.e. given a refutation in the tree with answer θ then $\forall(A_1,\ldots,A_n)\theta$ is a logical consequence of P.

- **Completeness** : Let θ be an answer for $P \cup \{G\}$, computed by the SLD procedure using C as "input" clause (driving clause), then there exists an answer σ computed by the "driven" method which is equivalent to θ, i.e. $G\sigma$ is a variant of $G\theta$. Such substitutions are said to be *dependent* on C.

- Every DM-computed answer uses clause C as "input" clause.

Thus, since we want to compute only the answers to $P \cup \{G\}$ that have a refutation containing, among the "input" clauses, the "driving" clause C, it is sufficient to inspect the tree defined by the DM-method.
The DM-method, uses an "ad hoc" computation rule that selects an atom A_i from the current goal, where A_i has to be resolved using information:

i) about the dependency relation among predicates in P, and
ii) about the "state" of the derivation.

To be more precise, since we want to build those refutations on the tree that use clause C as "input" clause, we select those atoms that can be directly resolved by C, or else atoms that, at some stage, generate subgoals that can be directly resolved by C. This strategy is applied until the clause C is used in the resolution, after that, the goal obtained at that stage becomes the goal of an SLD derivation procedure that uses a strategy whatsoever, e.g. the PROLOG's Leftmost Selection Rule.

In the following, in order to outline the method, we describe it for definite programs and for definite goals, while afterward we will extend it to normal goals, i.e. goals in which negative literals can occur, when discussing its application to an integrity constraint checking procedure.

Let us now consider the derivation tree that can be built by the above approach. The successful branches, whose associated substitutions are exactly those that can be obtained by using the clause C in the derivation, can be divided into two distinct phases:

1. - $G=G_0$, G_1, ..., G_i where each G_j $(0<j\leq i)$ is obtained by resolving in G_{j-1} an atom according to the DM-selection rule.

2. - G_{i+1},..., [] where G_{i+1} is the resolvent of G_i and C, and C is not used as "input" derivation clause in G_0 , G_1, ..., G_i.

2.2 Definitions

Let us give few definitions that are needed to define the method more precisely. For all those definitions which are not explicitly stated here we refer to [Lloyd 87].

Def. 1 Let P be a definite program and q a predicate symbol of P. The following sets can be inductively defined:

$SUCC^{0}(q) = \{ r \mid \exists C \in P$ such that $C= q(t_1,\ldots,t_n) \leftarrow L_1,\ldots, r(v_1,\ldots,v_m),\ldots,L_k$, and $r \neq q\}$

$SUCC^{i+1}(q) = \{ r \mid \exists C \in P$ such that $C= s(t_1,\ldots,t_n) \leftarrow L_1,\ldots, r(v_1,\ldots,v_m),\ldots,L_k$, $s \in SUCC^{i}(q)$ and $r \neq q$, $r \notin SUCC^{j}(q)$ $0 \leq j \leq i\}$

$SUCC(q) = \bigcup_{i \geq 0} SUCC^{i}(q)$

Since, SUCC (q) is contained in the set of predicates of P, it can be effectively computed in a finite number of steps, we need only to stop when a $SUCC^{j}(q)$ is found such that

$$SUCC^{j}(q) = SUCC^{j-1}(q)$$

Using SUCC (q) it is possible to give sufficient conditions to obtain the desired refutations: that is, all those refutations that use, among the input clauses, clause C. Such refutations are said "driven by C".

Def. 2 Let $C = A \leftarrow A_1,\ldots,A_n$ such that $A=p(t_1,\ldots, t_n)$:
head(C) = A; Body(C) = $A_1,\ldots,A_n$; predicate (A) = p.

Prop. 1. Let P be a definite program, C a clause of P and $G = \leftarrow A$ a definite goal.
If predicate(head(C)) $\notin$ SUCC(predicate(A)) every SLD derivation starting from G in P does not contain C.

proof: Straightforward from definition of SUCC.

Thus, given a goal, if we want to select all those derivations that use C as input clause it is necessary to expand all those atoms, A_i, in the goal such that predicate(head(C)) $\in$ SUCC(predicate(A_i)). If such an atom does not exist in G the derivation can fail because, from the above proposition, we know that no "interesting" derivations are obtainable from that goal.

Thus the idea of the algorithm is that of forcing, by means of an appropriate computation rule, the use of such atoms until C does appear as input clause, afterward the derivation may proceed in the usual way (i.e. by using the standard leftmost computation rule).

Def. 3. Let G be a goal and σ, θ two substitutions. σ, θ are equivalent iff $G\sigma$, $G\theta$ are variants.

Def. 4 (Driven Selection Rule, DSR)
Let P be a definite program, $G = \leftarrow A_1,\ldots, A_n$ a goal, C the driving clause, let q=predicate(head(C)) then

$$DSR(G) = \begin{cases} A_i \text{ if } q \in SUCC(predicate(A_i)) \text{ and } q \notin SUCC(predicate(A_j)) \text{ with } 0<j<i \\ A_i \text{ if } q = predicate(A_i) \text{ and } q \notin predicate(A_j) \text{ with } 0<j<i \\ \{\} \text{ otherwise .} \end{cases}$$

Let us now define the DM method.

The method takes a definite program P, a clause $C \in P$ and a definite goal G. The aim is to obtain all the correct answer substitutions for $P \cup \{G\}$ driven by C, that is all those refutations that use the clause C, at least once, as input clause.

The algorithm consists of the inductive construction of the following sets of goals:

Def. 5

$$\text{PHASE } I^{o} = \begin{cases} \{\} & \text{if DSR}(G) = \{\} \\ \{(G, \varepsilon)\} & \text{otherwise} \end{cases}$$

PHASE $I^{i+1} = \{(G_{k+1}, \theta_{k+1}) \mid (G_k, \theta_k) \in$ PHASE I^i with $G_k = \leftarrow A_1,\dots,A_n$, with n>0 such that:

(- DSR $(G_k) = A_j$ and predicate$(A_j) \neq$ predicate (C)

- $A \leftarrow B_1,\dots, B_m \in P \setminus \{C\}$ and $A_j\theta = A\theta$ with θ mgu.
- $\theta_{k+1} = \theta_k \circ \theta$
- $G_{k+1} = \leftarrow (A_1,\dots,A_{j-1}, B_1,\dots, B_m, A_{j+1},\dots,A_n\ )\theta)$

or • (-DSR $(G_k) = A_j$ and predicate(A_j) = predicate (C)

- $\theta_{k+1} = \theta_k$
- $G_{k+1} = G_k$) }

the construction terminates when PHASE $I^i = \{(G_k, \theta_k) \mid$ with $G_k = \leftarrow A_1,\dots,A_n$ and it exists an A_j s.t. predicate (C) = predicate $(A_j)\}$

We set PHASE I = PHASE I^i. ♦

Intuitively the construction of sets PHASE I^i corresponds to successive deductions in a partially computed SLD tree. Now deduction proceeds by solving the goals in PHASE I with the driving clause C to obtain the set PHASE II.

Def. 6

PHASE II = $\{(G_{k+1}, \theta_{k+1}) \mid (G_k, \theta_k) \in$ PHASE I with $G_k = \leftarrow A_1,\dots,A_n$, n>0 an A_j such that :

- head(C)$\theta = A_j\, \theta$ with θ mgu
- $\theta_{k+1} = \theta_k \circ \theta$
- $G_{k+1} = \leftarrow (A_1,\dots,A_{j-1}, \text{Body(C)}, A_{j+1},\dots,A_n\ )\theta\ \}$ .... ♦

The set PHASE II exactly contains the SLD tree nodes obtained after resolving with the input clause C. In order to obtain the complete refutations, deductions from these nodes have to be continued.
Thus, in order to obtain all the refutations, it is enough to expand each node in PHASE II by using SLD with the standard computation rule, leftmost, for example.

The DM method will give as a result the set of answer substitutions computed by the SLD on the goals in PHASE II, composed with the substitutions computed during PHASE I.

2.3 Completeness and correctness

In this section, we prove that the algorithm proposed is correct in the sense that each refutation in our method is an SLD refutation and it is complete in the sense that for every SLD refutation in which C appears (at least once) as input clause it exists a refutation in our method which gives an identical answer substitution.

Prop.2 (correctness)
Let P be a definite program, G a definite goal and C input clause. Then each answer substitution computed with the DM method is a correct answer substitution

Proof Straightforward; each refutation in the DM is a SLD refutation. ♦

Prop.3 (completeness)
Let P be a definite program, G: $\leftarrow A_1,\ldots,A_n$ a definite goal, and C input clause. Then for each SLD refutation of $P \cup \{G\}$ which uses C as input clause there exists an equivalent DM refutation.

Sketch of the proof
Let $G{=}G_0, \ldots, G_k, G_{k+1}, \ldots G_n = []$

$\theta_0 \quad \theta_k, \theta_{k+1} \quad \theta_n$

$c_0 \quad c_k, c_{k+1} \quad c_n$

be the SLD refutation, since it uses C as input clause there must exist a literal L_i in G_0 s.t. predicate(head(C)) $\in$ SUCC(predicate(L_i)), the DM method would have chosen such an L_i to proceed in the refutation. Now by the independence of the computation rule [th. 9.9, in Lloyd 87] the result follows.

♦

2.4 Normal goals: Two new computation rules

Let us now extend the approach to *normal* goals i.e. goals in which negative literals can occur.
Let us consider the following hypothesis:
- P is modified into P' either by adding or deleting a clause; then we want to compute all the correct answer substitutions for $P' \cup \{G\}$, related to a certain clause C, inspecting an inference structure which is smaller than the original SLDNF tree for $P' \cup \{G\}$.

Now in the case of normal goals we need two different driven selection rules instead of the one defined in the case of definite goals :

Def. 7 (Driven Selection Rule, DSRp)
Let P be a definite program, G = $\leftarrow L_1,\ldots, L_n$ a goal, C the driving clause, let q=predicate(head(C)) then

此为试读，需要完整PDF请访问：www.ertongbook.com

$$\mathrm{DSRp}(G) = \begin{cases} L_i \text{ if } L_i \text{ is a positive literal: } q=\mathrm{predicate}(L_i),\ q\neq\mathrm{predicate}(Lj) \text{ and} \\ \qquad\qquad q\notin \mathrm{SUCC}(\mathrm{predicate}(Lj) \text{ with } 0\leq j<i. \\ L_i \text{ if } L_i \text{ is a positive literal, } q\in \mathrm{SUCC}\ (\mathrm{predicate}(L_i)),\ q\notin \mathrm{SUCC}(\mathrm{predicate}(Lj)) \text{ with } 0<j<i. \\ \{\}\ \text{otherwise} \end{cases}$$

Def. 8 (Driven Selection Rule, **DSRn**)
Let P be a definite program, $G = \leftarrow L_1,\ldots, L_n$ a goal, C the driving clause, let q=predicate(head(C)) then

$$\mathrm{DSRn}(G) = \begin{cases} L_i \text{ if } L_i \text{ is a negative ground literale s.t. } q = \mathrm{predicate}(L_i) \text{ or } q \in \mathrm{SUCC}\ (\mathrm{predicate}(L_i)) \\ L_j \text{ if } L_j \text{ is a positive literal s.t. it contains at least a variable that also occurs in a negative} \\ \qquad \text{literal } L_k \text{ s.t. } q \in \mathrm{SUCC}\ (\mathrm{predicate}(L_k)) \text{ or } q = \mathrm{predicate}(L_k) \text{ and } 0<k<i. \\ \{\}\ \text{otherwise.} \end{cases}$$

In the next section we will show how to use these two computation rules to perform integrity constraint checking.

3. Applying DM resolution to constraint checking

In this section we show how it is possible to use the described driven method to obtain an efficient integrity constraint proof procedure which is correct and complete.

Let us assume that the database D, satisfies the integrity constraint IC before the update. In order to show that the IC is still satisfied,after the update, it has to be proved that the new database D' (to be precise the completion of D', comp(D')) is consistent with IC.

Roughly speaking the idea here is to consider integrity constraints formulas that are goals, i.e. $\leftarrow$ W, then use the DM with the update to D' as driving clause, if a finite failure is obtained then D' satisfies the integrity constraint otherwise the update violates D'.

Note that this approach to integrity constraint satisfaction has been proposed in [Sadri&Kowalski 87], and looks for a finite failure of D $\cup$ { $\leftarrow$ W} in order to establish consistency. Other approaches in the literature [Decker 86, Lloyd et al. 87, Asirelli et al. 88] use a specular notion of IC consistency preserving, namely that the IC has to be a logical consequence of the DB. In such approaches if D' preserves consistence, w.r.t. IC = W, then D $\cup$ { $\leftarrow$ W} has a refutation. The two approaches are equivalent for *complete* theories and when we restrict to the class of logic programs for which SLDNF is complete.

For a detailed presentation and comparison of the two approaches see [Sadri&Kowalski 87].

The idea of using the DM method consists of:
1) for updates that are *insertions*, select literals in $\leftarrow$ W according to **DSRp** in D'.
2) for updates that are *deletions*, select literals in $\leftarrow$ W according to **DSRn** in D'.

In fact, when updates are insertions, there may be branches in the derivation tree for D' $\cup$ { $\leftarrow$W} involving the inserted clause C, that were not in the derivation tree for D $\cup$ { $\leftarrow$ W}, if we assume that D$\cup$ { $\leftarrow$W} finitely fails, then D' $\cup$ { $\leftarrow$ W} may have a refutation on those new branches, that have to be checked for finite failure. On the other hand, the only literals in W that can have a refutation, depending on C are the positive ones, and those are the literals selected by **DSRp.**

When the update is a deletion, then there may be failure branches in the derivation tree for $D \cup \{\leftarrow W\}$ involving the deleted clause C, those branches now may become success branches in the derivation tree for $D' \cup \{\leftarrow W\}$, so they have to be checked for finite failure. On the other hand, the only literals in W that can now succeed, depending on the deletion of C are the negative ones, and those are the literals whose expansion is forced by using **DSRn**.

Let us now define the integrity verification method, splitting it in two parts: **VMDi** that deals with insertions and **VMDd** that deals with deletions.

Def. 9 VMDi

Let C be the inserted clause in D such that $D' = D \cup \{C\}$. Let $\leftarrow W$ be the IC.

1. Compute PHASE I on D' with $G = \leftarrow W$ and driving clause C, by using DSRp(G) instead of DSR(G).
2. Compute PHASE II. Let FINAL denote the set of SLDNF answer computed on $D' \cup \{G\}$ with $G \in$ PHASE II. If FINAL = {} **success** (D' satisfies the constraint $\leftarrow W$), otherwise **failure** (D' violates $\leftarrow W$).

VMDi is quite immediate: since we are dealing with an insertion, we know that if the constraint is violated after the update this can be caused, directly or indirectly, only because some positive literal contained in the constraint now succeed. In particular, this means that it must exist a positive literal, let it B, in W such that $\leftarrow B\sigma$ failed in D while in D' it has a refutation. Since all the new introduced derivations must involve the update, what we need to do is to select all the new branches in D' that contain C as input clause checking if a refutation exists and this is the case in which FINAL is not empty.

Def 10 (PHASE In)

$$\text{PHASE In}^{\circ} = \begin{cases} \{\} \text{ if DSRn } (G) = \{\} \\ \{(G, \varepsilon)\} \qquad \text{otherwise} \end{cases}$$

PHASE $\text{In}^{i+1} = \{(G_{k+1}, \theta_{k+1}) \mid (G_k, \theta_k) \in \text{PHASE I}^i$ with $G_k = \leftarrow L_1,\ldots,L_n$, with $n>0$ such that:

(- DSRn $(G_k) = A_j$ positive literal

- $A \leftarrow B_1,\ldots, B_m \in D'$ and $A_j\theta = A\theta$ with θ mgu.
- $\theta_{k+1} = \theta_k \circ \theta$
- $G_{k+1} = \leftarrow (A_1,\ldots,A_{j-1}, B_1,\ldots, B_m, A_{j+1},\ldots,A_n)\theta$)

or (- DSR $(G_k) = L_j$ negative ground literal

- $\theta_{k+1} = \theta_k$
- $G_{k+1} = G_k$) }

the construction terminates when given, q= predicate(head(C)) , all the negative literals $L_i \in G_{k+1}$ s.t. $q \in$ SUCC (predicate(L_i)) or q = predicate(L_i) are ground.

Let PHASE In denote the last computed PHASE In^{i+1}. ♦

PHASE In is needed in order to guarantee that the "interesting" negative literals to be selected become ground before selection, this corresponds to guarantee that our computation rule is *safe* [Lloyd 87]. Note that since D' is a definite program the negative literal that are considered are all and only those included in the IC.

Def. 11 VMDd

Let C be the deleted clause in D such that D'= D \ {C}. Let ← W be the IC.

1. Compute PHASE In on D' with G= ← W and driving clause C.

2. Let FINAL denote the set of SLDNF answers computed on D' ∪ {G} with G ∈ PHASE In. If FINAL = {} **success** (D' satisfies the constraint ← W), otherwise **failure** (D' violates ← W).

♦

Theorem 1 (Insertion Theorem VMDi)

Let C be the inserted clause in D such that D'= D ∪ {C}. Let ← W be the IC. Let us assume that D satisfies IC, i.e. D ∪ { ←W} finitely fails. Let SUCC be defined as in Sect. 2.2, and computed on D', then D' satisfies ←W iff VMDi terminates with **success**.

Sketch of the proof

→ VMDi terminates with **success** when FINAL = {}. This happens either when PHASE II = {} or when no refutation has been obtained from D' ∪ {G}, G∈ PHASE II.

All we need to prove to get the result is that PHASE II contains all the possible (partial) derivations tha contain C has input clause. This is straightforward from the results on the method in the case of positive programs and positive goals in sect. 2.

♦

Theorem 2 (Deletion Theorem VMDd)

Let C be the deleted clause in D such that D'= D \ {C}. Let ← W be the IC. Let s assume that D satisfies IC, i.e. D ∪ { ←W} finitely fails. Let SUCC be defined as in Sect. 2.2, and computed on D', then D' satisfies ←W iff VMDd terminates with **success.**

Sketch of the proof:

VMDd terminates with **success** if FINAL is empty . This happens either when PHASE In = {} or when no refutation has been obtained from D' ∪ {G}, G∈ PHASE In. All we need to prove to get the result is that PHASE In contains all the derivations that have to be checked. Since we have a deletion only a negative literal in IC may now be refuted, and this may cause IC violation. But PHASE In exactly contains all the goals in which the negative literals, whose proof uses C, are ground.

♦

It is straightforward to generalize the above definitions to the case of a generic transaction consisting of several insertions and deletions.

Examples

Example 1.

$A(x,y) \leftarrow D(x), C(y)$	$SUCC(A)= \{D,C,B,E\}$, $SUCC(D)= \{B,E\}$
$D(x) \leftarrow B(x)$	$SUCC(C)=SUCC(E)= SUCC(B)= \{\}$
$D(x) \leftarrow E(x)$	
$C(b)$	

$IC = \leftarrow A(x,y), \neg B(x)$ update = insert $E(d)$

DSRp $(IC) = A(x,y)$

PHASE $I^{\circ} = \{\leftarrow A(x,y), \neg B(x), \varepsilon\}$

DSRp $(\leftarrow A(x,y), \neg B(x)) = A(x,y)$

PHASE $I^1 = \{\leftarrow D(x), C(y), \neg B(x), \varepsilon\}$

DSRp $(\leftarrow D(x), C(y), \neg B(x)) = D(x)$

PHASE $I^2 = \{(\leftarrow B(x), C(y), \neg B(x), \varepsilon), (\leftarrow E(x), C(y), \neg B(x), \varepsilon)\}$

DSRp $(\leftarrow B(x), C(y), \neg B(x), \varepsilon) = \varnothing$

DSRp $(\leftarrow E(x), C(y), \neg B(x), \varepsilon) = \varnothing$

PHASE $I^3 = \{\}$

PHASE I = PHASE $I^2 = \{(\leftarrow B(x), C(y), \neg B(x), \varepsilon), (\leftarrow E(x), C(y), \neg B(x), \varepsilon)\}$

PHASE II = $\{\leftarrow C(y), \neg B(d), x/d\}$

$$\begin{tikzcd}
\leftarrow C(y), \neg B(d) \arrow[d, dash, "y/b"] \\
\leftarrow \neg B(d) \arrow[d, dash] \\
\square
\end{tikzcd}$$

FINAL = $\{y/b\}$

Thus the IC has been violated by the update.

Example 2.

$A(x) \leftarrow B(x), D(x)$ SUCC(A)= {B,D}, SUCC(C)= {F}

$C(x) \leftarrow F(x)$ SUCC(B)=SUCC(D)= SUCC(F)= {}

$B(b)$

$D(b)$

$F(b)$

$IC = \leftarrow \neg C(x), A(x)$ update = delete F(b)

DSRn (IC) = A(x)

PHASE $I_n^0 = \{((\leftarrow \neg C(x), A(x)), \varepsilon)\}$

DSRn $(\leftarrow \neg C(x), A(x)) = A(x)$

PHASE $I_n^1 = \{((\leftarrow \neg C(x), B(x), D(x)), \varepsilon)\}$

DSRn $(\leftarrow \neg C(x), B(x), D(x)) = B(x)$; **DSRn** $(\leftarrow \neg C(x), B(x), D(x)) = D(x)$

PHASE $I_n^2 = \{((\leftarrow \neg C(b), D(b)), x/b), ((\leftarrow \neg C(b), B(b)), x/b)\}$

FINAL ={ x/b}

thus the Integrity Constraint is violated.
On the contrary if, e.g., we consider update = delete D(b), at the first step **DSRn** (IC) = { }, thus getting FINAL = { } and concluding that the IC is still satisfied.

5 Conclusions

In this paper we have presented a method that allows the selection of those branches of a derivation tree that contain a specific clause as input clause. We have applied such method to integrity constraint checking but it seems useful in other application areas as well.

Despite to its simplicity in the case of definite goals and programs, when introducing negation problems arise. We restricted ourselves to definite programs and normal goals because in the case of integrity checking it seems quite a general framework since constraints act as goals in our method.
Note that if one makes the assumption that all the negative literals in the goal are ground, the verification method sensibly improves.

The presented examples serve as an explanatory framework rather than providing convincing elements with respect to the advantages of the method. In order to get such an evidence examples that give rise to a more complex search spaces are needed.

Much work has to be done to extend the approach to normal programs even if we foresee less advantages.

References

[Asirelli et al. 85] Asirelli, P., De Santis, M., Martelli, M., Integrity Constraints in Logic Data Bases, *Journal of Logic Programming*, Vol. 2, n. 3, Oct. 1985.

[Asirelli et al. 88] P. Asirelli, P. Inverardi, A. Mustaro "Improving Integrity constraint Checking in Deductive Databases" , International Conference on Database Theory , 29 Agoust, 2 September 1988, Bruges, Belgium, LNCS 326.

[Decker 86] Decker, H., Integrity Enforcement on Deductive Databases , *1st International Conference on Expert Database Systems,* Charleston, S.C., 1986, pp. 271-285.

[Lloyd 87] Lloyd, J. W., *Foundations of logic programming*, Symbolic Computation Series, Springer, 1987, Second Edition.

[Lloyd et al. 87] Lloyd, J. W., Sonenberg, E.,A., Topor, R. W., Integrity Constraint Checking in Stratified Databases, J. Logic Programming 4 (4): 331-343, 1987.

[Sadri&Kowalsky 87] Sadri, F., Kowalsky , R. A., A Theorem-Proving Approach to Database Integrity, in: J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufman, Los Altos, 1987.

此为试读，需要完整PDF请访问： www.ertongbook.com