MANAGEMENT OF STRATIFIED DATABASES

Krzysztof R. Apt and Jean-Marc Pugin[1]

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-87-41 November 1987

[1]BULL Research Center, P.C 58 A 14-A.I. Division, 68 route de Versailles, 78430 Louveciennes, France.

Management of Stratified Databases

Krzysztof R. Apt
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
and
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188
USA

Jean-Marc Pugin
BULL Research Center
P.C 58 A 14-A.I. Division
68 route de Versailles, 78430 Louveciennes
France

We propose here a knowledge based management system supporting immediate visualization and simulation facilities in presence of non-monotonic reasoning. It is based on a special class of indefinite deductive databases, called stratified databases, introduced in APT, BLAIR and WALKER [ABW] and VAN GELDER [VG], in which recursion "through" negation is disallowed.

A stratified database has a natural model associated with it which is selected as its intended meaning. The main technical problem addressed here is the task of maintaining this model. To solve it we refine the ideas present in the works of DOYLE [D] and DE KLEER [dK] on belief revision systems. We also discuss the implementation issues and the trade-offs involved.

Note: Work partially supported by the ESPRIT project 415. Preliminary version of this paper appeared as APT and PUGIN [AP].

1. INTRODUCTION

1.1. Objectives

The aim of this paper is to propose a knowledge based management system (KBMS in short) whose main characteristics are: use of incomplete information, immediate visualization of modifications, generation of explanations, simulation and "undo" capabilities. Our proposal has a clear semantics allowing us to account for the use of incomplete information in an interactive environment. We believe that due to the above features our proposal can be used as a system for interactive problem solving and decision making. The framework in which we carry out our investigations is that of deductive databases, or more generally rule based programming. While proposing such a system we have in mind the following objectives:

1. Use of incomplete information.

It is well understood by now that monotonic reasoning is not sufficient to adequately describe human style of reasoning, mainly because the assumption that all needed information is available, is unrealistic.

So a major feature of a realistic KBMS should be its ability to deal with *incomplete information*. In particular, such a system should be able to draw conclusions in the absence of some informations, and to withdraw them when some information contradictory with the assumptions becomes available.

2. Permanent interaction between the system and the user.

A dynamic character of the knowledge makes it desirable to provide an interactive KBMS whose models are generated in an incremental fashion through an interaction with the user. Our definition of "interaction" allows both additions and deletions of the clauses. This gives us a means to model simulation.

3. Simulation facilities.

An important aspect of decision making systems is their ability to simulate some representation of the world. It is possible in some advanced systems to use a $WHAT-IF$ function in order to analyze the consequences of a modification, and subsequently to return to the previous state with an "undo" facility if needed. The use of the $WHAT-IF$ function can be viewed as a *look ahead* facility allowing us to analyze one of the next possible states of the system without commiting oneself to this state.

We provide this facility by offering a very general definition of interaction. In our proposal the user can not only (hypothetically) modify the actual situation by changing the set of facts but also temporarily change actual laws represented by the set of rules. When a given in advance condition (an *integrity constraint*) does not hold, the system will refuse to perform the modification. We think that this new type of interaction with a KBMS helps in an incremental problem solving because it allows the user to investigate consequences of some of the laws before deciding which ones to choose.

4. Generation of explanations.

Explanations are essential when one deals with interactive problem solving. When a desired fact is generated, we would like to be able to provide the reasons for which this fact holds. In our proposal such an explanation consists of the rules which triggered this fact.

The above considerations constitute a part of specifications of a new type of facility called *Logical Spreadsheet* described in CRAS, LECONTE and PUGIN [CLP]. These specifications form a basis for an internal project at the BULL Research Center which aims at producing a prototype of the Logical Spreadsheet. Logical Spreadsheets are intended to serve as an advanced interface in an environment in which rule based programming or object oriented programming is used.

1.2. Means

Our approach is based on logic programming. A natural representation for handling incomplete information is the one in which negative hypotheses are allowed in rules. A negative hypothesis, say $\neg A$, should then be interpreted as "if so far A cannot be confirmed" which models the non-monotonic character of reasoning. These and other aspects of negation were intensively studied in the framework of logic programming. Use of negation increases the expressiveness of the syntax (see CHANDRA and HAREL [CH]) but leads to several fundamental difficulties (see e.g. SHEPHERDSON [S1,S2]. In particular, it is not clear what is the intended declarative meaning of the program. Recently, APT, BLAIR and WALKER [ABW] and independently VAN GELDER [VG] proposed a simple solution to the latter problem obtained by imposing a restriction on the syntax, namely by disallowing recursion "through" negation. This class of programs, called *stratified programs*, forms a simple generalization of a class of database queries introduced in CHANDRA and HAREL [CH]. It admits a simple declarative semantics in the form of a particular minimal model, which enjoys several natural properties (see APT, BLAIR and WALKER [ABW] and VAN GELDER [VG], LIFSCHITZ [L] and PRZYMUSINSKI [Pr]).

Dynamic aspects of non-monotonic reasoning were studied by DOYLE [D], DE KLEER [dK] and others in the form of Truth Maintenance or Belief Revision Systems - a class of A.I. programs which maintain consistency by manipulating a set of supports used in conditional proofs. In DOYLE [D] when an inconsistency is detected a special mechanism is invoked to alter the supports associated with the conditionally derived facts. In DE KLEER [dK] in case of detection of inconsistency, the inconsistent part of the system (set of assumptions) is identified and associated contexts are removed.

In this paper we combine the declarative and dynamic aspects of non-monotonic reasoning by studying the management of stratified databases, i.e. deductive databases which when seen as a logic

program are stratified. As their intended meaning we choose the above mentioned model.

This representation is powerful enough to meet our objectives. First, the interaction with the user is modeled by means of updates, or more generally, by transactions. Secondly, the immediate visualization is viewed as the task of generating the new model of the modified stratified database. Finally, the simulation facility is embodied in the possibility of issuing a transaction "converse" to the previous one which effectively "undoes" the previous transaction. The solution proposed automatically embodies a possibility of generating an explanation for the newly obtained facts.

Processing of updates and transactions results in the maintenance of the intended model in absence of complete information. This requires the use of powerful capabilities to compute the "new" model using the "old" one. It is precisely the main technical point addressed and solved in this paper.

1.3. Plan of the paper

As stratified programs form a basis for our proposal, in section 2 we recall their definition and main results concerning their semantics.

In section 3 we formulate the problem of maintenance in terms of computing the "new" model from the "old" one. We also introduce the notion of *migration*, which is a useful parameter to compare solutions to this problem. The non-monotonicity is reflected in the fact that insertions can lead to deletions and vice-versa. To handle this problem we track dependencies between the facts from the generated model and relations used in their derivations. These dependencies are attached to the facts and called *supports*.

In section 4 we analyze a spectrum of solutions to the maintenace problem resulting from a different choice of supports. All these solutions consist of two phases, namely the removal and the addition phase.

In section 5 we study a different type of solutions - those in which the stratification is used to propagate modifications through the strata and in which the removal and addition phases are interleaved.

In section 6 we extend one of the solutions proposed in section 5 to the case of transactions, i.e. sequences of updates.

In section 7 we show how the solution given in section 6 can be modified so that integrity constraints can be checked during the construction of the new model.

In section 8 we propose an efficient implementation of the algorithms given in sections 5, 6 and 7. We also indicate there that this implementation automatically takes care of the problem of generating the explanations for the newly derived facts.

Finally, in section 9 we briefly discuss related work in the area of deductive databases and Artificial Intelligence.

2. STRATIFIED PROGRAMS - AN OVERVIEW

We recall here briefly the results of APT, BLAIR and WALKER [ABW] which form a basis for this work.

2.1. Definitions

Throughout this paper we assume a fixed first order language L. An *atom* is a formula of L which is of the form $p(\bar{t})$ where p is a relation symbol of arity n and $\bar{t}$ is a sequence of terms of L of the length n. If all terms in $\bar{t}$ are ground (i.e. variable free) then we call $p(\bar{t})$ a *fact*. A *literal* is an atom (also called a *positive literal*) or its negation (called a *negative literal*).

A clause is a formula of L of the form

$$A \leftarrow L_1,...,L_n$$

where $n \geq 0$, A is an atom and $L_1,...,L_n$ are literals. A is called the *conclusion* of the clause and

$L_1,...,L_n$ its *body*. If the body of the clause is not empty (i.e. $n>0$) then we call the clause a *rule*.

A relation symbol occurs *positively* in a clause if it appears in its positive literal. In particular the relation appearing in the conclusion of the clause occurs in it positively. A relation symbol occurs *negatively* in a clause if it appears in its negative literal.

Finally, a *logic program* (or just a *program*) is a finite, non-empty set of clauses of L. Given a program, a *definition* of a relation symbol is the set of clauses of the program using it in its conclusion.

Given a program P, we denote by B_P the set of facts of L whose relation symbol appears in P. B_P is called the *Herbrand base* associated with P. A *Herbrand interpretation of P* is a subset of B_P. Given a relation p and a Herbrand interpretation M, $[p]_M$ stands for the meaning of p in M, i.e. the set of facts of the form $p(\bar{t})$ true in M.

Given a logic program P we define its *dependency graph* D_p by putting:

(r,q) belongs to D_p iff there is a clause in P using r in its conclusion and q in its body.

We then say that r *refers to* q. If q occurs positively in the body, then we call the arc (r,q) *positive*. If q occurs negatively in the body, then we call the arc (r,q) *negative*. Note that an arc can be both positive and negative because q can appear both positively and negatively in a (not necessarily the same) rule using r in its conclusion.

Now, following APT, BLAIR and WALKER [ABW], a logic program is called *stratified* if no cycle in its dependency graph contains a negative arc (intuitively: there is no recursion "passing through" a negation). Equivalently, a program P is stratified if there is a partition (where P_1 can be empty) $P = P_1 \cup \cdots \cup P_n$, called a *stratification* of P, such that for $i = 1,...,n$

a) if a relation symbol occurs positively in a clause in P_i then its definition is contained in $\cup_{j\leq i} P_j$,

b) if a relation symbol occurs negatively in a clause in P_i then its definition is contained in $\cup_{j<i} P_j$.

Each P_i is called a stratum.

This definition implies that a stratum R of a stratified program is a union of the definitions of some relation symbols such that if a relation symbol occurs negatively in a clause from R then its definition is disjoint with R. In general there is more than one way to stratify a program. A stratification $P_1 \cup ... \cup P_n$ of P is *maximal* if no stratum in it can be further decomposed into different strata.

Given a stratum P and a set of facts M from L, we denote by SAT(P,M) - *the saturation of M by P* - the set of facts obtained by closing the set M under the clauses of P. Given a stratification $P_1 \cup \cdots \cup P_n$ of P we put:

$$M_1 = SAT(P_1, \emptyset),$$

...

$$M_n = SAT(P_n, M_{n-1}),$$

$$M_P = M_n,$$

and call M_P the standard model of the program P.

In general, $SAT(P,M)$ can depend on the order of rule application, but this is not the case when P is a stratum. The actual implementation of the saturation process is discussed in detail in section 8.

Let M be a Herbrand model of a program P. M is called *minimal* if no proper subset of it is a model of P. M is called *supported* if for every element A of it there exists an explanation for it in the form of an instance of a clause of P whose body is true in M and whose conclusion is A.

2.2. Results

Using some general results on fixpoints of non-monotonic operators on complete lattices the following properties of the model M_P were proved in APT, BLAIR and WALKER [ABW].

THEOREM: Let P be a stratified program. Then

i) M_P does not depend on the stratification of P,
ii) M_P is a minimal model of P,
iii) M_P is a supported model of P,
iv) there is an equivalent definition of M_P which makes use iteratively smallest models as follows:

$$M_1 = \cap\{M : M \text{ is a supported model of } P_1\},$$

$$M_2 = \cap\{M : M \text{ is a supported model of } P_2 \text{ and } M \cap B_{P_1} = M_1\}$$

...

$$M_n = \cap\{M : M \text{ is a supported model of } P_n \text{ and } M \cap B_{P_1 \cup \cdots \cup P_{n-1}} = M_{n-1}\},$$

v) M_P is a model of comp(P), Clark's [C] completion of P,
vi) there is an (ineffective) backchaining interpreter for P using the negation as failure rule and loop checking (but working only with fully instantiated clauses) which tests for the membership in M_P. This interpreter becomes effective when P is function-free. □

Other properties of M_P were proved independently by VAN GELDER [vG]. LIFSCHITZ [L] showed that M_P can also be defined using the circumscription method of MCCARTHY [MC]. PRZYMUSINSKI [Pr] generalized the above results by introducing an (ineffective) form of resolution that allows to test for membership in M_P. He also introduced a notion of a *perfect model* and showed that every stratified program has exactly one perfect model, namely M_P. This provides in our opinion an ample evidence that M_P is a natural model for a stratified program P.

The choice of a Herbrand model as the semantics of P can be viewed as a compact representation of the intended meaning of P. In the model only (atomic) facts are explicitly recorded. This allows us to answer directly a query about validity of a particular fact. However, a query about validity of a first order formula has to be computed using the standard definition of truth.

3. STRATIFIED DATABASES AND THE MAINTENANCE PROBLEM

3.1. Definition

From now on we assume that the first order language L has no function symbols and only finitely many, but at least one constant.

Following GALLAIRE, MINKER and NICOLAS [GMN]) we define a *deductive database* as a function-free logic program in the above language L augmented by the usual particularization axioms defining uniquely its domain and the equality predicate. P is divided into

i) a set of facts defining extensional relations (Extensional Database),
ii) a set of clauses defining intentional relations, all of them different from extensional relations (Intentional Database).

Moreover, each clause in P is *range restricted* which means that every variable which appears in a conclusion of the clause also appears in its body. Note that this implies that clauses of a deductive database can be divided into facts and rules.

In addition a deductive database contains a finite set of integrity constraints. These are first order formulas which are required to be continuously true in the sense described in the next subsection. Now, by a *stratified database* we mean a deductive database built from a stratified logic program.

A stratified database P has as its intended meaning the standard model M_P. When maintaining P

two representation possibilities arise:
i) *explicit representation* consisting of P and M_P,
ii) *implicit representation* consisting just of P.

Which alternative is more attractive depends on the application. Alternative i) is more appropriate when trying to support immediate visualization and simulation facilities. Also i) is more interesting when dealing with frequent queries and infrequent updates.

Consequently, we choose, similarly as NICOLAS and YAZDANIAN [NY] for the case of definite deductive databases (i.e. those in which use of negation in the clauses is disallowed), the explicit representation.

As we shall soon see, we shall actually maintain an enrichment of M_P in which each fact from M_P is tagged with some additional information.

It is worthwhile to note that alternative ii) leads to difficult problems concerning an efficient implementation of queries which only recently have been solved in a satisfactory way - see BALBIN, PORT and RAMAMOHANARAO [BPR] and KERISIT, LESCOEUR, ROHMER and ROUCAIROL [KLRR].

3.2. Maintenance

The maintenance problem can be viewed as a task of processing supplementary information. To this purpose we first define the notion of an update. By an *update* of a stratified database P we mean a clause deletion or insertion. We require that in the case of the insertions
i) no constant outside of L is introduced,
ii) the inserted clause is range restricted ,
iii) the resulting database P' remains stratified.

Updates can be divided into fact insertions and deletions and rule insertions and deletions because all clauses are assumed to be range restricted.

The maintenance problem can now be formulated as follows:

given an update of a stratified database P yielding P' compute the intended meaning $M_{P'}$ of P' making use of the already existing model M_P of P. If this update leads to a model which does not satisfy the integrity constraints, then a failure should be reported.

Thus we require that the integrity constraints continuously hold in the intended model of the stratified database. Until section 7 we ignore the issue of the integrity constraints checking and concentrate on the problem of processing the updates. The computation of $M_{P'}$ using M_P is closely related to the issue of *dependency-directed backtracking* discussed in STALLMAN and SUSSMAN [SS]. In general, $M_{P'}$ will be neither a superset or subset of M_P.

Consider for example the stratified database

$$PODS = \{submitted(1),...,submitted(\ell),accepted(n_1),...,accepted(n_k),$$
$$\{rejected(x) \leftarrow \neg accepted(x)\}$$

where $k, \ell \geqslant 1$ and for $i = 1,...,k$ $1 \leqslant n_i \leqslant \ell$ holds.

Its model M_{PODS} consists of all facts already present in PODS together with the set of facts rejected(i) for $i \in$ Failure $= \{1,...,\ell\} \setminus \{n_1,...,n_k\}$.

Now an insertion of the fact accepted(m) where $m \in$ Failure leads to a new database PODS' with the following associated model

$$M_{PODS'} = M_{PODS} \setminus \{\text{rejected(m)}\} \cup \{\text{accepted(m)}\}.$$

Similarly, a deletion of the fact accepted(n_j) where $1 \leqslant j \leqslant k$ leads to a new database PODS" with the following associated model

$$M_{PODS''} = M_{PODS} \setminus \{\text{accepted}(n_j)\} \cup \{\text{rejected}(n_j)\}.$$

Thus to compute the new model $M_{P'}$, it is in general necessary to remove some facts from M_P and also add some other facts.

In the next two sections we study the maintenance problem in the case of updates. Then we study this problem for the more general case of transactions which are finite sequences of updates.

3.3. The STRATIFY procedure

Let now P be a stratified database. Assume a given maximal stratification of P with the corresponding sequence of models $M_1,...,M_m = M_P$. Note that in case of insertions a new stratum can be created and in case of deletions a stratum can "disappear". However, the resulting maximal stratification $P_1 \cup \cdots \cup P_n$ of P' is such that one of the following conditions holds

i) exactly one stratum of P' differs from a stratum of P,
ii) this stratification is obtained by removing one stratum from the initial stratification of P,
iii) this stratification is obtained by adding one stratum to the initial stratification of P , say as the last one.

In one case the solution to the maintenance problem is trivial. Consider an update consisting of a deletion of a clause which results in removing the highest stratum P_{n+1} from P. Then the model $M_{P'}$ simply consists of M_P with the last "layer" $M_{P_{n+1}} \setminus M_{P_n}$ removed. Therefore, in the subsequent considerations we do not consider this case. This allows us to introduce the following definition.

If an update results in a deletion of an "intermediate" stratum from P, we say that it *refers to* the next stratum in the stratification of P'. Otherwise, we say that an update *refers to* a stratum R from the stratification P' if the definition of the relation appearing in the conclusion of its clause is contained in R.

We assume that the maximal stratification $P_1 \cup \cdots \cup P_n$ of P' is computed in the procedure STRATIFY(P, u, P',i) where P is the original stratified database, u is an update, P' is the resulting stratified database, and i such that u refers to P_i. Note that then $P_1,..., P_j$ for $j < i$ are initial strata in the original stratification of P.

We can assume that the conditions i) - iii) from the previous subsection 3.2 are checked in this procedure and a failure is reported if one of them is not met.

To compare solutions to the maintenance problem we concentrate on the issue of a *migration of facts* - a phenomenon consisting of an erroneous removal of a fact from the model. In such case, this fact has to be added back to the model. Different solutions to the maintenance problem can be compared in terms of the amount of migration caused.

While searching for good solutions to the maintenance problem it makes sense to strike a balance between the minimization of migration and the cost of bookkeeping involved. We think that the solution proposed in section 5 achieves this compromise because of an efficient implementation proposed in section 8. The bookkeeping consists of a maintenance of supports attached to the facts present in the model. These supports will allow us to detect which facts should be removed from the model after an insertion or deletion.

4. Two phase solutions

In this section we present various solutions to the maintenance problem in the case of updates. They differ in the form of supports chosen. For pedagogical reasons we order these solutions in such a way that each of them builds upon deficiences found in the previous one. No attempt is made at proposing efficient implementations of these solutions. Throughout this section P is a given stratified database.

4.1. Static solution using the dependency graph

This is perhaps the simplest solution and usually the most inefficient one. In this solution no supports are attached to the facts in the model. Instead, the dependency graph is used. For each relation p of P, let $Pos(p)$ stand for the set of relations of P from which p depends through an even number of negations and $Neg(p)$ stand for the set of relations of P from which p depends through an odd number of negations. Thus

$Pos(p) = \{q$: there exist relations $p_1 = p,...,p_n = q$, such that for all $i<n$ (p_i,p_{i+1}) belongs to D_p and the number of negative arcs among them is even $\}$,

$Neg(p) = \{q$: there exist relations $p_1 = p,...,p_n = q$, such that for all $i<n(p_i,p_{i+1})$ belongs to D_p, and the number of negative arcs among them is odd $\}$.

Note that $Pos(p)$ and $Neg(p)$ need not be disjoint; $Pos(p) \cup Neg(p)$ is the set of all relations in P from which p depends.

We use here the notations *Pos* and *Neg* to indicate the nature of dependencies between the meaning of relations in the model. If r depends on p then a modification of p through an update can influence the meaning of r in the new model. The form of this influence implies the type of dependency of r on p. Suppose that an increase of p leads to some decrease of r. Then p belongs to $Neg(r)$. Suppose that a decrease of p leads to some decrease of r. Then p belongs to $Pos(r)$.

The following lemma formalizes this observation.

Let $[p]_M$ stands here for the meaning of the relation p in the Herbrand model M.

LEMMA 1.

Suppose that $p(\bar{t})$ *is a fact.*

i) *Let* $P' = P \cup \{p(\bar{t})\}$.

If not $([r]_{M'} \subseteq [r]_M)$ *then* p *belongs to* $Neg(r)$.

ii) *Let* $P' = P \setminus \{p(\bar{t})\}$.

If not $([r]_{M'} \subseteq [r]_M)$ *then* p *belongs to* $Pos(r)$.

Proof idea. By an induction on the index of the stratum which contains the definition of the relation r. □

Thus in the case of an insertion of a fact about p, only relations r, for which p belongs to $Neg(r)$, can decrease and in the case of a deletion of a fact about p only relations r, for which p belongs to $Pos(r)$, can decrease. We use these observations in the procedures below.

Fact insertion:

INSERT $(p(\bar{t}))$:

1) STRATIFY(P,$INSERT(p(\bar{t}))$,P',i);
2) remove from M_P all facts $r(\bar{s})$ such that p belongs to $Neg(r)$;
   (these facts all belong to $M_P \setminus M_{i-1}$)
3) add $p(\bar{t})$ and call the resulting set of facts M;
4) compute the sequence

$$M'_i = SAT(P_i,M),$$

...

$$M'_n = SAT(P_n,M'_{n-1})$$

and put $M_{P'} = M'_n$.

Rule insertion:

INSERT $(p(\bar{x})\leftarrow L_1,...,L_k)$:
1) STRATIFY(P,$INSERT(p(\bar{x})\leftarrow L_1,...,L_k)$,$P'$,$i$);
2) recompute the sets $Pos(r)$ and $Neg(r)$ for $r \equiv p$ and all relations which depend on p;
3) perform step (2) of the fact insertion. Call the result M;
4) perform step (4) of the fact insertion.

Fact deletion:

DELETE$(p(\bar{t}))$:
1) STRATIFY(P,$DELETE(p(\bar{t}))$,P',i);
2) remove from M_P all facts $r(\bar{s})$ such that p belongs to $Pos(r)$;
(these facts all belong to $M_P \setminus M_{i-1}$)
3) remove $p(\bar{t})$ and call the resulting set of facts M;
4) perform step (4) of the fact insertion.

Rule deletion:

DELETE $(p(\bar{x})\leftarrow L_1,...,L_k)$:
1) STRATIFY(P,$DELETE(p(\bar{x})\leftarrow L_1,...,L_k)$,$P'$,$i$);
2) recompute the sets $Pos(r)$ and $Neg(r)$ for $r \equiv p$ and all relations r which depend on p;
3) perform step (2) of the fact deletion and call the resulting set of facts M;
4) perform step (4) of the fact insertion.

In all four procedures during the removal phase we take a "pessimistic" view and delete facts taking into account *exclusively* the dependencies recorded in the dependency graph. Clearly certain facts will then be subject to migration.

EXAMPLE 1. Let

$$CONF = \{submitted(1),...,submitted(\ell), late(\ell+1),$$
$$accepted(x)\leftarrow submitted(x),\neg rejected(x),$$
$$accepted(\ell+1)\}$$

where $\ell \geq 1$.

Then M_{CONF} consists of all facts already present in $CONF$ together with the fact: $accepted(1),...,accepted(\ell)$.

However, after the insertion of the fact $rejected(\ell+1)$ in $CONF$ we should not remove the fact $accepted(\ell+1)$ from the model. In this case the static solution leads to a migration of the fact $accepted(\ell+1)$. □

Thus the static analysis using the dependency graph can provide dependencies which are not used during the construction of the model. This problem can be overcome by constructing the dependencies in a dynamic fashion.

NOTE. The presence of facts in a given program like $accepted(\ell+1)$ in $CONF$ above cannot be discovered through the analysis of the dependency graph of the program but it still can be viewed as a part of a static analysis. This idea might "save" certain facts like $accepted(\ell+1)$ from migration. However, this solution falls down when some trivial derivations for certain facts are used instead of

asserting them.

4.2. Dynamic solution using Pos and Neg sets

We now maintain M_P by computing the *Pos* and *Neg* sets dynamically during the construction of the model, i.e. during the saturation process iterated through the strata. This leads to a better solution because the *Pos* and *Neg* sets are computed taking into account the dependencies *actually* used and not the *potential* ones. However, as we shall soon see, the use of negative literals complicates the issue. Each fact in the model M_P has a support in the form of *Pos* and *Neg* sets attached to it. Their actual form depends on the way the saturation process is implemented.

We are interested in keeping the *Pos* and *Neg* sets small. In such a way less facts will be deleted during the removal phase in each of the above four procedures. To this purpose for each fact we just record the dependencies found during a deduction of this fact. These *Pos* and *Neg* sets should not be changed unless a smaller pair of them is found during another deduction of the fact. This idea leads to the following construction.

Suppose that during the model construction a fact $p(\bar{t})$ is deduced by an application of a clause $p(\bar{x})\leftarrow L_1,\ldots,L_k$ with some substitution making every literal L_i ground. Among those ground literals, let $q_1(\bar{s}_1),\ldots,q_i(\bar{s}_i)$ be the positive ones and $\neg r_1(\bar{t}_1),\ldots,\neg r_j(\bar{t}_j)$ the negative ones. As the positive ground literals $q_1(\bar{s}_1),\ldots,q_i(\bar{s}_i)$ already belong to the constructed part of M_P, they have the corresponding sets $Pos_1,\ldots,Pos_i$ and $Neg_1,\ldots,Neg_i$ attached to them.

We form the *Pos* and *Neg* sets attached to $p(\bar{t})$ as follows:

$$Pos := Pos_1 \cup \cdots \cup Pos_i \cup \{q_1,\ldots,q_i\},$$

$$Neg := Neg_1 \cup \cdots \cup Neg_i \cup \{r_1,\ldots,r_j\}.$$

If $p(\bar{t})$ is already present in the model, we keep its old pair of *Pos* and *Neg* sets unless the new pair is pairwise smaller than the old one. In that case the new pair is preferable. As before, the *Pos* and *Neg* sets need not be disjoint.

Insertions and deletions are performed analogously as in 4.1 but now using the above *Pos* and *Neg* sets attached to all facts of the model. For example, in step (1) of the fact insertion concerning $p(\bar{t})$ we now remove from M_P all facts $r(\bar{s})$ whose *Neg* set contains the relation p and then add $p(\bar{t})$ with a support consisting of empty *Pos* and *Neg* sets.

Unfortunately this solution is incorrect.

EXAMPLE 2. Let $P = \{p_1\leftarrow\neg p_0, p_2\leftarrow\neg p_1, p_3\leftarrow\neg p_2\}$.
Then $M_P = \{p_1,p_3\}$.

After an insertion of the fact p_0 we get a new database P' with a model $M_{P'} = \{p_0,p_2\}$. However, the removal of the fact p_3 from M_P is not captured by the solution proposed above.

Indeed, the *Neg* set attached to p_3 in the model M_P equals $\{p_2\}$ and the crucial (negative) dependency of p_3 from p_0 is not recorded. Similarly, a deletion of the fact p_0 leads to the model $M_P = \{p_1,p_3\}$. However, the removal of the fact p_2 from $M_{P'}$ is not captured by the proposed solution. In this example, all constructed *Pos* sets are empty. □

To resolve these difficulties in the case of negative hypotheses we keep track of their static dependencies, as well. The actual construction and form of these supports remains almost the same. What changes is their use during the updates. Given the above mentioned deduction of $p(\bar{t})$ we form the *Pos* and *Neg* sets attached to it by putting

$$Pos := Pos_1 \cup \cdots \cup Pos_i \cup \{q_1,\ldots,q_i\} \cup \{-r_1,\ldots,-r_j\},$$

$$Neg := Neg_1 \cup \cdots \cup Neg_i \cup \{+r_1,\ldots,+r_j\}.$$

During the updates we compute the *actual* form of the supports by interpreting the signed relations

as follows:

$$Pos' = \{q : q \in Pos\} \cup Neg(r_1) \cup \cdots \cup Neg(r_j)$$

where for $k = 1,...,j$ $-r_k \in Pos$,

$$Neg' = \{q : q \in Neg\} \cup Pos(r_1) \cup \cdots \cup Pos(r_j) \cup \{r_1,...,r_j\}$$

where for $k = 1,...,j$ $+r_k \in Neg$.

$Neg(r)$ and $Pos(r)$ refer here of course to the sets defined in section 4.1, i.e. to the static dependencies.

Intuitively, Pos' is the set of relation symbols used positively in the found derivation of $p(\bar{t})$ and Neg' is the set of relation symbols used negatively in the found derivation of $p(\bar{t})$. Each derivation of $p(t)$ provides a different pair of Pos' and Neg' sets. Only one of such pairs is kept in this solution.

The details of the insert and delete procedures are the same as before. The above modification restores correctness of this solution. The following lemma states the relevant property of the Pos' and Neg' sets.

LEMMA 2.

Suppose that $p(\bar{t})$ is a fact.

i) *Let $P' = P \cup \{p(\bar{t})\}$.*

Suppose that $r(\bar{s})$ belongs to $[r]_{M_P} \setminus [r]_{M_{P'}}$, i.e. that $r(\bar{s})$ was removed from the model M_P. Then p belongs to the Neg' set associated with $r(\bar{s})$ in the model M_P.

ii) *Let $P' = P \setminus \{p(\bar{t})\}$.*

Suppose that $r(\bar{s})$ belongs to $[r]_{M_P} \setminus [r]_{M_{P'}}$, i.e. that $r(\bar{s})$ was removed from the model M_P. Then p belongs to the Pos' set associated with $r(\bar{s})$ in the model M_P.

Proof idea. By an induction on the index of the stratum which contains the definition of the relation r. □

In contrast to lemma 1, lemma 2 refers to sets Pos' and Neg' whose form depends on the actual form of the saturation procedure computing the sets $SAT(P,M)$.

In the case of the database P from example 2 the facts of the model are generated only in one possible sequence. The resulting Pos' and Neg' sets coincide with their static counterparts. The following example shows an interest in keeping a pair of smaller supports if a choice arises.

EXAMPLE 3. Let

$$CONGRESS = \{submitted(1),...,submitted(\ell),$$
$$accepted(x) \leftarrow submitted(x), \neg rejected(x),$$
$$accepted(\ell) \leftarrow submitted(\ell)\}.$$

Suppose now that the fact accepted(ℓ) is first deduced by the first rule. Then the associated Pos and Neg sets have the following form:

$$Pos = \{submitted, -rejected\} \text{ and } Neg = \{+rejected\}.$$

If the second rule is applied we obtain another pair of Pos and Neg sets associated with the fact $accepted(\ell)$:

$$Pos = \{submitted\} \text{ and } Neg = \varnothing.$$

Clearly, the latter pair is preferable because an insertion of a fact $rejected(i)$ will not lead then to a migration of the fact $accepted(\ell)$. □

Though this solution leads to smaller migration sets than the previous one it can still lead to some

inaccuracies. The major reason is that only one support is kept for each deduced fact. Thus the maintained information can be incomplete. Consider the following example.

EXAMPLE 4. Let

$$MEET = \{submitted(1),...,submitted(\ell),$$
$$in-program-committee(name_1),...,in-program-committee(name_9),$$
$$author(m_1,1),...,author(m_t,\ell),$$
$$accepted(x) \leftarrow submitted(x),\ \neg rejected(x),$$
$$accepted(y) \leftarrow author(x,y),\ in-program-committee(x)\}$$

where $\ell \geq 1$ and $\{name_1,...,name_9\} \subseteq \{m_1,...,m_t\}$.

Then M_{MEET} consists of all facts already present in P together with the facts accepted(1),...,accepted(ℓ).

Suppose now that the fact author($name_2$,a) is in *MEET*. Then after the insertion of the fact rejected(a) we should not remove the fact accepted(a) from the model. However, if for the fact accepted(a) the support Pos = {submitted,− rejected}, Neg = {+ rejected} is initially produced, it will lead to its migration. Here the second possible support Pos = {author, in-program-committee}, $Neg = \emptyset$ is preferable with respect to this update but this support is not kept. □

To take care of this type of situations we should maintain supports in the form of *Pos* and *Neg* sets *for each derivation* of a fact, and thus maintain supports not in the form of sets but rather sets of sets. This observation leads us to the following solution.

4.3. Dynamic solution using Pos and Neg sets of sets

The sets *Pos* and *Neg* will now be sets of sets of relations. Intuitively, when a fact $p(\bar{t})$ has a set $Pos = \{A_1,...,A_k\}$ associated with it, it means that for each set A_j a derivation of $p(\bar{t})$ has been found in which exactly all relations from A_j are negated an even number of times. Similarly with the *Neg* set.

Let $B_1,...,B_k$ be non-empty sets of sets. We put:

$$B_1 \oplus \cdots \oplus B_k = \{A_1 \cup \cdots \cup A_k : A_i \in B_i \text{ for } i = 1,...,k\}$$

During the model construction in the case discussed in the beginning of the previous subsection *Pos* and *Neg* sets are now updated as follows:

$$Pos := Pos \cup (Pos_1 \oplus \cdots \oplus Pos_i) \oplus \{\{q_1,...,q_i,-r_1,...,-r_j\}\}$$
$$Neg := Neg \cup (Neg_1 \oplus \cdots \oplus Neg_i) \oplus \{\{+r_1,...,+r_i\}\}$$

with *Pos* and *Neg* initialised to the empty set.

Thus each time a new deduction of a fact has been found, its *Pos* and *Neg* sets are updated as stated above. If a fact has a trivial deduction, i.e. it is asserted, its *Pos* and *Neg* sets will both have the empty set as an element. Similarly as in the previous subsection we might be interested in keeping only "small" supports. That is, we might remove an element A from *Pos* (or *Neg*) each time a proper subset of it has been added to *Pos* (or *Neg*).

Because the supports have now a different structure, the removal phase in each of the four procedures will be different. Intuitively, a fact should now be removed from the model only if *all* elements of its support "fail". More precisely, in accordance with the previous solution we first put for an element A which belongs to *Pos*

$$A' = \{q : q \in A\} \cup Neg(r_1) \cup \cdots \cup Neg(r_j)$$

where for $k = 1,...,j$ $-r_k \in A$, and for an element A which belongs to *Neg*

$$A' = \{q : q \in A\} \cup Pos(r_1) \cup \cdots \cup Pos(r_j)$$

where for $k = 1,...,j$ $+r_k \in A$.

Then in the case of an insertion of a fact $p(\bar{t})$ we proceed as follows during the removal phase:

for each element $r(\bar{s})$ of the model **do**
1 remove from its *Neg* set all elements A such that p belongs to A';
2 if the *Neg* set becomes empty remove $r(\bar{s})$ from the model.
od.

Thus a "failure" of an element of a support means here that p belongs to it.

An analogous action is taken during the removal phase in other three procedures.

To see an improvement over the previous solution reconsider the program from example 4. During the construction of the model M_{MEET} both supports of the fact *accepted*(a) will be kept. Thus the *Pos* and *Neg* sets associated with *accepted*(a) will have the following form:

$$Pos = \{\{submitted, -rejected\}, \{author, in-program-committee\}\}$$

$$Neg = \{\{+rejected\}, \varnothing\}.$$

Now, after the insertion of the fact *rejected*(a) we see that *rejected* belongs to $\{+rejected\}' = \{rejected\}$, so the *Neg* set associated with *accepted*(a) becomes $\{\varnothing\}$. Since it is not empty, the fact *rejected*(a) is not removed from the model, as desired.

5. Incremental solutions

So far we discussed solutions to the maintenance problem which consisted of two phases: the *removal phase* during which some facts were deleted, followed by the *addition phase* during which some facts were inserted. We now present another type of solutions in which the removal and the addition phases are alternated. This will lead to solutions with smaller migration and among others will obviate the need for the static information in the supports.

Informally, this form of solutions can be described as follows. Consider an update u of P resulting in a stratified database P' with a maximal stratification $P_1 \cup \cdots \cup P_n$. The original model M_P of P can be decomposed into a sequence of *layers* $N_1 = M_1$, $N_2 = M_2 \setminus M_1$,..., $N_n = M_m \setminus M_{m-1}$ where $|m-n| \leqslant 1$, with each M_i corresponding to a stratum in the original maximal stratification of P.

Suppose that u refers to P_i. Then to construct $M_{P'}$ we first consider the modification of the layer N_i. This leads to deletions and insertions inside N_{i+1} which in turn leads to deletions and insertions inside N_{i+2}, etc. This form of solutions thus produces a *cascade effect*.

5.1. Auxiliary procedures

To describe this process we shall introduce three procedures. We describe them for the form of supports used in the second dynamic solution i.e. in subsection 4.3. It is clear how to modify them for the case of supports used in the first dynamic solution.

Assume a given maximal stratification $P_1 \cup ... \cup P_n$ of a stratified database P'.

1) The SATURATE procedure

The purpose of the procedure *SATURATE*(*Stratum*, B) is to compute the saturation of the current version of the model using all clauses the *Stratum*, and update during this computation the *Pos* and *Neg* sets of sets attached to every derived fact. The result of this saturation becomes a new version of the model. B is the set of relations which increased.

SATURATE(Stratum,B):

a) Compute the set SAT(*Stratum*,*M*) where *M* is the current version of the model and during this computation update the *Pos* and *Neg* sets attached to the derived facts. This time these sets are constructed as follows.
Suppose that a fact $p(\bar{t})$ is deduced by means of a clause such that $q_1,...,q_i$ are all relations which appear positively in its body and $r_1,...,r_j$ are all relations which appear negatively in its body. Then the sets *Pos* and *Neg* associated with $p(\bar{t})$ are updated as follows:

$$Pos := Pos \cup \{\{q_1, \ldots, q_i\}\},$$

$$Neg := Neg \cup \{\{r_1,...,r_j\}\}$$

with *Pos* and *Neg* initialised to the empty set.

b) *B* is the set of relations to which new facts where added in step (a).

2) The REMOVEPOS procedure

Let *B* be the set of relations defined in the strata below the current one, which decreased during the construction of the new model carried out so far. Their decrease can affect the supports of the facts from the layer corresponding with the current stratum and, in particular, can lead to a decrease of some of the relations defined in the current stratum. The purpose of the *REMOVEPOS*(*Stratum*,*B*,*C*) procedure is to compute this modification using the *Pos* part of the supports. *C* is the set of relations defined in the current stratum which get decreased.

REMOVEPOS(Stratum,B,C):

Consider the elements of $M = M_i \setminus M_{i-1}$, where Stratum = P_i.
$C := \emptyset$;
for each element $p(\bar{t})$ of *M* **do**
remove from its *Pos* set all sets *A* such that $A \cap B \neq \emptyset$;
if the *Pos* set becomes empty **then** remove $p(\bar{t})$ from M_P and set $C := C \cup \{p\}$ **fi**
od.

3) The REMOVENEG Procedure

Let *B* be the set of relations defined in the strata below the current one, which increased during the construction of the new model carried out so far. Their increase can affect the supports of the facts from the layer corresponding with the current stratum and, in particular, can lead to a decrease of some of the relations defined in the current stratum. The purpose of the *REMOVENEG*(*Stratum*,*B*,*C*) procedure is to compute this modification using the *Neg* part of the supports. *C* is the set of relations defined in the current stratum which get decreased.

REMOVENEG (Stratum,B,C):

Consider the elements of $M = M_i \setminus M_{i-1}$, where Stratum = P_i.
$C := \emptyset$;
for each element $p(\bar{t})$ of *M* **do**
remove from its *Neg* set all sets *A* such that $A \cap B \neq \emptyset$;
if the *Neg* set becomes empty **then** remove $p(\bar{t})$ from M_P and set $C := C \cup \{p\}$ **fi**
od.

5.2. Algorithms

We now present the update algorithms in the case of incremental solutions. They use the procedures SATURATE, REMOVEPOS and REMOVENEG defined above.

Fact insertion:

INSERT $(p(\bar{t}))$:
Initialize:

$DEC := \emptyset$; $INC := \emptyset$;
STRATIFY(P, $INSERT(p(\bar{t}))$, P', i);
if $p(\bar{t}) \in M_P$ **then**
modify the support of $p(\bar{t})$ as follows:
$Pos := Pos \cup \{\emptyset\}$;
$Neg := Neg \cup \{\emptyset\}$
else continue := **true**
fi;

Propagate:

if continue **then**
while $i \neq n+1$ **do**
Stratum := P_i;
REMOVEPOS(Stratum,DEC,DECPOS):
REMOVENEG(Stratum,INC,DECNEG);
SATURATE(Stratum,ADD);
DEC := DEC ∪ DECPOS ∪ DECNEG;
INC := INC ∪ ADD;
$i := i+1$
od
fi.

Note that when the fact $p(\bar{t})$ is already in the model its support is modified and no further action is taken. Note also that the *Propagate* part is executed exactly when the control passes through the **else** part of the *Initialize* part. We preferred here to isolate the *Propagate* part in order to use it in other algorithms.

In the above algorithm, DEC (INC) is the set of relations which were decremented (incremented) so far during the construction of the model. Maintaining the sets DEC and INC allows us to use the current form of supports. Note that these supports are now "one level deep" as opposed to the previous form in which practically whole proof trees were maintained. This difference can be also found in the approaches of Doyle [D] and De Kleer [dK]. In Doyle [D] the latter type of supports is used whereas De Kleer [dK] uses the previous form which allows him to maintain several contexts at the same time.

An improvement of the above algorithm can be obtained by taking into account the structure of each stratum. When proceeding through the **while** loop one can skip the strata in which no relation depends from a relation in the set DEC ∪ INC.

To see how this version of fact insertion leads to a smaller migration than the algorithm given in subsection 4.3, consider the database $P = \{r \leftarrow p,\ q \leftarrow r,\ q \leftarrow \neg p\}$. Then $M_P = \{q\}$. INSERT (p) when computed using the previous version leads to the removal of q, followed by the insertion of p and r and finally the insertion of q. In the above version the removal of q does not take place.

Rule insertion:

INSERT $(p(\bar{x}) \leftarrow L_1,...,L_k)$:
Initialize:

DEC := $\varnothing$; INC := $\varnothing$;
STRATIFY(P, *INSERT*$(p(\bar{x}) \leftarrow L_1,...,L_k)$, P', i);
continue := **true**;

Propagate.

Note that contrary to the case of fact insertion at least one iteration of the loop in the *Propagate* part is performed. An improvement of the above algorithm can be obtained by terminating this loop when after the first iteration the SATURATE procedure produces no new facts, that is when both DEC and INC remain empty.

Fact deletion:

DELETE$(p(\bar{t}))$:
Initialize:

DEC := $\varnothing$; INC := $\varnothing$;
STRATIFY(P, *DELETE*$(p(\bar{t}))$, P', i);
modify the support of $p(\bar{t})$ as follows:
Pos := *Pos* $\setminus \{\varnothing\}$; *Neg* := *Neg* $\setminus \{\varnothing\}$
if both *Pos* and *Neg* sets become empty **then**
    remove $p(\bar{t})$ from M_P;
    DEC := DEC $\cup$ $\{p\}$;
    continue := **true**
fi;

Propagate.

Note that when the fact $p(\bar{t})$ remains in the model, its supports are modified and no further action is taken. Indeed, the model remains then the same and other supports do not change.

Rule deletion:

DELETE $(p(\bar{x}) \leftarrow L_1,...,L_k)$:
Let $q_1,...,q_i$ be all relations which appear positively in the body $L_1,...,L_k$ and let $r_1,...,r_j$ be all relations which appear negatively in this body.

Initialize:

DEC := ∅; INC := ∅; removed := **false**;
STRATIFY(P, *DELETE*($p(\bar{x})\leftarrow L_1,...,L_k$), P', i);
for each element $p(\bar{t})$ of $M_i \setminus M_{i-1}$ **do**
  remove from its *Pos* set the set $\{q_1,...,q_i\}$
  and from its *Neg* set the set $\{r_1,...,r_j\}$ if
  both sets are effectively present;
  if both *Pos* and *Neg* sets become empty **then**
    remove $p(\bar{t})$ from M_P;
    removed := **true**
  fi
od;
if removed **then** DEC := DEC ∪ $\{p\}$; continue := **true** **fi**;

Propagate.

Note that in the *Initialize* part an attempt is made to identify the facts of the form $p(\bar{t})$ which were deduced in only one way, namely by means of the rule $p(\bar{x})\leftarrow L_1,...,L_k$.

6. Transaction processing

So far we have dealt with the processing of updates. In this section we consider a more general situation, namely that of transactions.

Following Lloyd, Sonenberg and Topor [LST] by a *transaction* we mean a finite sequence of updates. We can assume without loss of generality that in any transaction we do not have insertions and deletions of the same clause. We can now order a sequence of updates forming a transaction in such a way that those refering to lower strata appear first.

More precisely, we can order these updates in such a way that the values i returned by the procedure STRATIFY from section 3 when applied to these updates form a weakly increasing sequence. In order to be able to reuse this procedure we now add at the end of its body the assignment $P := P'$.

We now propose an algorithm showing how to process a transaction. It builds upon the incremental solution to the update processing proposed in the previous section. We assume that a transaction is ordered in the way explained above. Given a stratified database P with the model M_P and a transaction let P' be the resulting stratified database with the maximal stratification $P_1 \cup ... \cup P_n$.

Consider a stratum P_i and the corresponding part $M = M_i \setminus M_{i-1}$ of the model. We first introduce the MODIFY procedure whose purpose is to process the changes within M resulting from updates referring to stratum P_i. These changes consist in general of

i) removal of some facts,
ii) addition of some facts,
iii) resulting modification of the sets DEC and INC,
iv) modification of some supports.

This procedure has the following form.

MODIFY (Stratum,DEC,INC):
Consider the updates referring to the stratum *Stratum*. Perform their *Initialize* parts in any order but with the initial assignments DEC := ∅; INC := ∅ and the assignment continue := **true** everywhere deleted.

Now, the following algorithm is used to process a transaction, where i_0 is the smallest value returned by the procedure STRATIFY applied to the updates forming the transaction.

DEC := ∅; INC := ∅; $i := i_0$;
while $i \neq n+1$ **do**
    Stratum := P_i;
    MODIFY(Stratum,DEC,INC);
    REMOVEPOS(Stratum,DEC,DECPOS);
    REMOVENEG(Stratum,INC,DECNEG);
    SATURATE(Stratum,ADD);
    DEC := DEC ∪ DECPOS ∪ DECNEG;
    INC := INC ∪ ADD;
    $i := i+1$
od

This algorithm is more efficient than the one resulting from a one by one processing of the updates forming the transaction. Indeed, only one pass through the strata is made in it and all modifications are treated in a cumulative fashion.

7. Integrity constraints checking

Similarly as in Lloyd, Sonenberg and Topor [LST], by an integrity constraint we mean a first order formula in the language of the considered stratified database. Assume a stratified database P with a finite set of integrity constraints $F_1,...,F_k$.

Our intention is to check whether after processing a transaction leading from P to a new stratified database P', the new model $M_{P'}$ satisfies all formulas $F_1,...,F_k$. The simplest solution is to evaluate each of these formulas in $M_{P'}$. This however, does not take into account the fact that all $F_1,...,F_k$ were satisfied in the old model M_P.

We propose now a solution which allows us to identify a subset of $F_1,...,F_k$ which needs to be checked. Moreover, in this solution it is not necessary to wait until the construction of the new model is completed to evaluate each of the "suspected" integrity constraints.

Thus the construction of the new model will be aborted as soon as an integrity constraint is identified which does not evaluate to true.

Consider a first order formula F and its conjunctive normal form W. We say that a relation symbol occurs *positively* in F if it occurs in a positive literal in W. We say that a relation symbol occurs *negatively* in F if it occurs in a negative literal in W. Note that a relation symbol can occur both positively and negatively in a formula.

Consider now a stratifed database P and a transaction, and let P' be the resulting stratifed database with the maximal stratification $P_1 \cup \cdots \cup P_n$. We say that a formula F in the language of P' *refers to* a stratum P_i if the definitions of all relation symbols appearing in F are contained in $\cup_{j<i} P_i$.

Consider now a formula F refering to a stratum P_i. We can evaluate truth of F in the new model $M_{P'}$ once in the construction of this model, using the algorithm given in section 6, stratum P_{i+1} has been reached.

Moreover, such an evaluation of F is unnecessary if none of the relations appearing positively in F appears in the set *DEC*, and none of the relations appearing negatively in F appears in the set *POS*. Indeed, in that case F is true in $M_{P'}$ because it is true in M_P. This follows from the fact that truth of a formula in a Herbrand model is uniquely determined by the meaning of the relations appearing in this formula and from the following straightforward lemma.

Lemma 3.
Let M and M' be two Herbrand models and F a formula. Suppose that
i) *for all relations r appearing positively in F*, $[r]_M \subseteq [r]_{M'}$,

ii) *for all relations r appearing negatively in F*, $[r]_{M'} \subseteq [r]_M$.
Then F is true in M iff it is true in M'. □

We can apply this lemma here because once in the algorithm given in section 6 a new stratum has been reached, *DEC* (*INC*) includes the set of relations defined in the previous strata which were decremented (incremented) so far during the construction of the model.

Once a constraint does not evaluate to true, the construction of the new model is aborted. To reconstruct the old model it is enough to process a transaction "reverse" to the previous one (that is the one in which deletions are replaced by insertions and vice versa) and stop once the stratum has been reached at which the construction of the new model had been aborted. Indeed, the effect of both construction on the layers, which were taken care of, is nil, because both transactions cancel each other.

Finally, we offer the following improvement upon the previous method of identifying when an integrity constraint does not need to be verified. When evaluating an integrity constraint F in a (computed fragment of a) new model, we attach to it a support containing the information which relation symbols from each conjunct of the conjunctive normal form were used during this evaluation.

Similarly as in section 4.2 such a support consists of a set *Pos* of relations which appear positively in F and were used in this evaluation, and a set *Neg* of relations which appear negatively in F and were used in the evaluation. Then a constraint does not need to be evaluated if *Pos* is disjoint with *DEC* and *Neg* is disjoint with *INC*. Each time a constraint is evaluated anew, the sets *Pos* and *Neg* are computed anew.

8. Implementation issues

In this section we study the problem of an efficient implementation of the algorithms proposed in the previous sections. We concentrate on the incremental solutions in which supports consist of *Pos* and *Neg* sets of sets of relations.

8.1. Implementation of supports

There is an obvious dependence between the support of a fact from the model and the set of clauses which triggered this fact during the construction of the model. This suggests an efficient implementation in which the support of a fact consists of the set of pointers to the clauses which triggered this fact. Note that this implementation of supports automatically takes care of the problem of generating explanations for the newly derived facts. Indeed, the clauses derived from the support can be viewed as an explanation for the presence of the fact in the model.

We now explain how supports are maintained and used under this representation. To this purpose we explain their use in the algorithms proposed.

1) The SATURATE procedure.
Each time a fact is deduced during the construction of the model, a pointer to the last clause applied is added to the support of this fact.

2) The REMOVEPOS procedure.
Consider a set B of relations and an element $p(\bar{t})$ of M. All clauses in whose body a relation from B appears positively are removed from the support of $p(\bar{t})$. If the support becomes empty, $p(\bar{t})$ is removed from the model.

3) The REMOVENEG procedure.
Consider a set B of relations and an element $p(\bar{t})$ of M. All clauses in whose body a relation from B appears negatively are removed from the support of $p(\bar{t})$. If the support becomes empty, $p(\bar{t})$ is

removed from the model.

4) Fact insertion - the *Initialize part.*
If $p(\bar{t})$ is in M_P, then a pointer to itself is added to the support of $p(\bar{t})$.

5) Fact deletion - the *Initialize part.*
A pointer to itself is removed from the support of $p(\bar{t})$. If the support becomes empty, $p(\bar{t})$ is removed from the model.

6) Rule deletion - the *Initialize part.*
Consider an element $p(\bar{t})$ from $M_i \setminus M_{i-1}$. The deleted rule is removed from its support. If the support becomes empty, $p(\bar{t})$ is removed from the model.

8.2 Implementation of the SATURATE procedure.
As stated in section 2 the set SAT(P,M) for a stratum P of a stratified program and a set of facts M does not depend on the order of rule application. To see this, first note that relations negated in the hypotheses do not appear in the conclusions of rules from P. Thus their meaning remains fixed throughout the saturation process. This implies that the rules of P form a monotonic production system and the desired independency follows by a general result proved in COUSOT [Co].

We exploit this independency by making use of an efficient implementation of the saturation process proposed in ROHMER, LESCOEUR and KERISIT [RLK] for the case of definite deductive databases. This algorithm is called there the *delta driven mechanism*, and was first implemented in the framework of a relational production system in PUGIN [Pu]. It was also introduced in BANCILHON [B], where it is called *semi-naive evaluation.*

Informally, each rule when fired produces an increase (delta) of the relation in the conclusion of the rule. When this increase is non-empty all rules using this relation in a body can be fired. The process stops when all increases are empty.

More formally, this algorithm has the following form. Let $Q_1,...,Q_k$ be the predicates corresponding to the meanings of the relations $p_1,...,p_k$ occuring in P and M. Each rule r_i in P induces a mapping f_i from $Q_1,...Q_k$ to Conc(i), where Conc(i) is the predicate associated with the relation used in the conclusion of the rule r_i. This mapping is obtained by translating the rule into an expression of the relational algebra. Let $f_1,...,f_m$ be the mappings obtained. The algorithm has the following form:

add to M all facts from from P;
for $j := 1$ **to** k **do** $Q_i := [p_i]_M$;
$\Delta Q_i := Q_i$;
$\Delta\Delta Q_i := \varnothing$
od;
repeat
for $j := 1$ **to** k **do if** $\Delta Q_j \neq \varnothing$ **then**
for $i := 1$ **to** m **do**
$\Delta\Delta\ Conc(i) := f_i\ (Q_i, \ldots, \Delta Q_j, \ldots, Q_k) \cup \Delta\Delta\ Conc(i)$
od
od;
for $j := 1$ **to** k **do** $\Delta Q_j := \Delta\Delta Q_j \setminus Q_j$;

$Q_j := Q_j \cup \Delta Q_j$;
$\Delta\Delta Q_j := \varnothing$

od

until $\bigcup_{j=1}^{k} \Delta Q_j = \varnothing$

The above algorithm computes SAT(P,M). However, in our setting we also need to maintain supports attached to the facts produced. These facts are generated in chunks of the form ΔΔ Conc(i). Each of them is produced by one rule. Adding now to the support of each fact in ΔΔ Conc(i) a pointer to this rule r_i, we obtain a refinement we need to implement the SATURATE procedure.

8.3 Discussion

The supports constructed in subsection 4.2 and 4.3 use the supports already attached to individual facts derived from the body of the rule applied. To maintain them, each newly derived fact has to be handled individually. Thus the delta driven mechanism which produces new facts in chunks cannot be applied. This shows that from the implementation point of view the solution proposed in section 5 is clearly preferable.

Note however that in general there is a trade-off between an efficient choice of the supports and the minimization of the migration. Indeed, to maintain supports efficiently they should be kept small. But then each fact will be more often subject to migration.

One might consider a different form of supports in which not relations (or pointers to the clauses) but facts used in the deductions are recorded. This would be clearly preferable from the point of view of minimization of migration. In fact, this form of supports combined with an appropriate type of a saturation procedure keeping all possible *original* deductions would lead to a solution with no migration.

This solution could be of interest in the case of Artificial Intelligence applications where typically few facts and many rules are used.

However, this choice is less attractive in the case of database applications. First, use in the supports of pointers to the rules instead of facts, allows us to use the delta driven mechanism based on relational algebra operators to implement the saturation process. Secondly, the computation costs incurred in the task of analyzing all possible deductions is clearly too prohibitive to be of practical interest when many facts are present.

9. Related work

Deductive databases:

Nicolas and Yazdanian [NY] consider the maintenance problem for definite deductive databases. Absence of negation considerably simplifies the issue. Lassez, McAloon and Port [LMP] address the problem of interactive construction of the intended model of a stratified database in case of propositional programs, concentrating on the complexity issues. Their definition of interaction does not allow deletions of clauses and does not include the integrity constraints checking. Lloyd, Sonenberg and Topor [LST] study the problem of integrity constraint checking in stratified databases using constructions somewhat related to our formation of *Pos* and *Neg* sets. In their approach Clark's [C] completion is used as the intended semantics of the database. Topor and Sonenberg [TS] consider the problem of domain independent queries in stratified databases.

Non-monotonic Reasoning:
Doyle [D] introduces the class of justification-based Truth Maintenance Systems and studies them both from a theoretical and practical point of view. De Kleer [dK] and Martins and Shapiro [MS] introduce (we use here the original term of De Kleer) the class of Assumption-based Thruth Maintenance Systems. De Kleer gives a new, elegant notion of consistency by introducing the multiple context framework instead of using the classical scheme in which only one consistent context is selected and used by the maintenance system. In both papers the notion of *selective backtracking* in case of detection of inconsistency is studied. These issues were subsequently studied in other frameworks, for example in Shmueli et al. [STZE] for the case of PROLOG.

REFERENCES

[ABW]K.R. APT, H. BLAIR, and A. WALKER, *Towards a Theory of Declarative Knowledge,* in: Proc. Workshop on Foundations of Deductive Databases and Logic Programming, Washington D.C. pp. 546-629, 1986.

[AP]K.A. APT and J.M. PUGIN, *Maintenance of Stratified Databases viewed as a Belief Revision System,* in: Proc. 6th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems, 1987.

[BPR]I. BALBIN, G.S. PORT and K. RAMAMOHANARAO, *Magic set computation for stratified databases,* Department of Computer Science, The University of Melbourne, Technical Report No. 87-3, 1987.

[B] F. BANCILHON, *Naive Evaluation of Recursively Defined Relations,* MCC Technical Report No. DB-4-85, 1985.

[CH]A. CHANDRA, and D. HAREL, *Horn Clause Queries and Generalizations,"* Journal of Logic Programming, vol. 1, pp. 1-15, 1985.

[C] K. CLARK, *Negation as failure,* in: Logic and Databases, H. Gallaire and J. Minker (Eds.), Plenum Press, New York, pp. 293-322, 1978.

[Co] P. COUSOT, *Asynchronous Iterative Methods for Solving a Fixed Point System of Monotone Equations in a Complete Lattice,* Technical Report No. 88, L.A. 7, Univ. Scientifique et Medicale de Grenoble, 1977.

[CLP]J.Y. CRAS, M. LECONTE and J.M. PUGIN, *Specifications du tableur logique,* Bull Research Center, Louveciennes, France, Technical Report, 1987.

[D] J. DOYLE, *A Truth Maintenance System,* Artificial Intelligence 12, pp. 231-272, 1979.

[GMN]H. GALLAIRE, J. MINKER, and J.M. NICOLAS, *Logic and Databases: A Deductive Approach,* ACM Computing Surveys, pp. 153-185.

[VG]A. VAN GELDER, *Negation as Failure Using Tight Derivations for General Logic Programs,* in: Proc. Third IEEE Symposium don Logic Programming, Salt Lake City, Utah, 1986.

[KLRR]J.M. KERISIT, R. LESCOEUR, J. ROHMER and G. ROUCAIROL, *The Alexander Method - an Efficient Way for Handling Deduction on Databases,* Bull Research Center, Louveciennes, France, Technical Report no. 87-015, 1987.

[dK]J. DE KLEER, *An Assumption-Based Truth Maintenance System,* Artificial Intelligence 28, pp. 127-162, 1986.

[LMP]C. LASSEZ, K. MCALOON and G.S. PORT, *Stratification as a Tool for Interactive Knowledge Base Management,* in: Proc. 4th International Conference on Logic Programming, The MIT Press, Cambridge, Mass., 1987.

[L] V. LIFSCHITZ, *On the Declarative Semantics of Logic Programs with Negation,* in: Proc. Workshop on Foundations of Deductive Databases and Logic Programming, Washington D.C. pp. 420-432, 1986.

[LST] J.W. LLOYD, E.A. SONENBERG, and R. TOPOR, *Integrity Constraint Checking in Stratified Databases,* Technical Report 86/5, Dept. of Computer Science, Univ. of Melbourne, 1986.

[MC]J. MCCARTHY, *Circumscription - A Form of Non-monotonic Reasoning,* Artificial Intelligence 13, pp. 295-323, 1980.

[MS]J.P. MARTINS, and S.C. SHAPIRO, *Reasoning in Multiple Belief Spaces,* in: Proc. IJCAI-83, pp. 370-373, 1983.

[NY]J.M. NICOLAS, and K. YAZDANIAN, *An Outline of BDGEN: A Deductive DBMS,* in Proc. IFIP-83, pp. 711-717, 1983.

[Pu] J.M. PUGIN, *Boum: Manual de reference et d'utilisation,* Bull Research Center, Louveciennes, France, Technical Report, 1984.

[Pr] T. PRZYMUSINSKI, *On the Semantics of Stratified Deductive Databases,* in: Proc. Workshop on Foundations of deductive Databases and Logic Programming, Washington D.C. pp. 433-443, 1986.

[RLK]J. ROHMER, R. LESCOEUR and J.M. KERISIT, *The Alexander Method, a Technique for the Processing of Recursive Axioms in Deductive Databases,* New Generation Computing vol. 4, No.3, pp.273-285,1986.

[S1] J.C. SHEPHERDSON, *Negation as Failure: A Comparison of Clark's Completed Database and Reiter's C.W.A.,* Journal of Logic Programming N 1, pp. 51-81, 1984.

[S2] J.C. SHEPHERDSON, *Negation as Failure. II,* Journal of Logic Programming, N 3, pp. 185-202, 1985.

[SS] R.M. STALLMAN, and G.J. SUSSMAN, *Forward Reasoning and Dependency-Directed backtracking in a System for Computer-Aided Circuit Analysis,* Artificial Intelligence 9, pp. 135-196, 1977.

[TS] R. TOPOR, E.A. SONENBERG, *On Domain Independent Databases, in: Proc. Workshop on Foundations of deductive Databases and Logic Programming, Washington D.C. pp. 403-419, 1986.*

[STZE]O. SHMUELI, S. TSUR, H. ZFIRA, and R. EVER-HADANI, *Dynamic Rule Support in Prolog,* in: Expert Databases Systems (L. Kerchberg, ed.), The Benjamin/Cummings Publishing Co., Menlo Park, pp.247-270, 1986.