MAINTENANCE OF STRATIFIED DATABASES VIEWED AS A BELIEF REVISION SYSTEM+

Krzysztof R. APT*
Laboratoire d'Informatique
Ecole Normale Supérieure
45 rue d'Ulm, 75230 PARIS
and
LITP, Université Paris 7
2 Place Jussieu, 75251 PARIS

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

ABSTRACT

We study here declarative and dynamic aspects of non-monotonic reasoning in the context of deductive databases. More precisely, we consider here maintenance of 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 maintenance problem for these databases is complicated because insertions can lead to deletions and vice versa.

To solve this problem we make use of the ideas present in the works of Doyle [D] and de Kleer [dK] on belief revision systems. We offer here a number of solutions which differ in the amount of static and dynamic information used and the form of support introduced. We also discuss the implementation issues and the trade-offs involved.

1. INTRODUCTION

Use of incomplete information, for example in the case of hypothetical reasoning or in real time applications in which missing data has to be faced with, leads to non-monotonic reasoning. Depending on the application domain differents solutions to non-monotonic reasoning may arise and be needed. The framework in which we carry out our investigations is that of deductive databases, or more generally rule based programming.

+ Work partially supported by the ESPRIT project 415.
* First author's address after March 1, 1987 : CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands.

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 hypothetical character of reasoning. These and other aspects of negation were intensely 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] and Gallaire, Minker and Nicolas [GMN]. 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, admits a simple declarative semantics in the form of a particular minimal model, which enjoys several natural properties (see [ABW], [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 [D] when an inconsistency is detected a special mechanism is invoked to alter the supports associated with the conditionally derived facts. In [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 maintenance 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.

The non-monotonicity is reflected by the fact that insertions can lead to deletions and vice versa. To handle this problem we track dependencies between facts present in the model and relations used in their derivations. These dependencies can be either statically derived from the dependency graph or dynamically computed during the construction and subsequent modifications of the model. Depending which form of dependencies is used we obtain a

different solution to the maintenance problem. We offer here various solutions which rely successively on more dynamic information. We also study the implementation issues concluding on a version which admits a simple and efficient implementation. Finally we point out the trade-offs involved.

While the idea of using supports attached to facts present in the model directly relates to the work of [D] and [dK] the difference lies in the way they are constructed and used. In fact, in the context of deductive databases the issue is maintenance of the intended declarative meaning and not maintenance of consistency. This leads to differents considerations and different solutions.

2.STRATIFIED PROGRAMS - AN OVERVIEW

We recall here briefly the results of Apt, Blair and Walker [ABW] which form a basis for this work.

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 a conclusion and q in a hypothesis. To an arc (r,q) which belongs to D_P we attach an information whether a reference of r to q is positive (i.e q occurs positively in the hypotheses) or negative (i.e q occurs negatively in the hypotheses). In the first case we speak of a positive arc, and in the second case of a negative arc. An arc can be both positive and negative because a reference of r to q can be both positive and negative (not necessarily in the same rule).

Now, following [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) $P = P_1 \cup ... \cup P_n$ 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 $\cup P_j$ for $j \leq i$.

b) if a relation symbol occurs negatively in a clause in P_i then its definition is conttained in $\cup P_j$ for $j < i$.

Recall that a definition of a relation symbol is the set of clauses using it in its conclusion. Given a logic program P and a set of ground atoms (or facts) M, we denote by SAT(P,M) - the saturation of M by P - the set of ground atoms obtained by closing the set M under the rules of P. Given a stratification $P_1 \cup ... \cup P_n$ 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) depends on the order of rule application, but this is not the case here. The actual implementation of the saturation process is discussed in detail in section 5. Also, 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.

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. Recall that B_P denotes the set of ground atoms in the language of the program P, i.e the Herbrand base associated with P.

Using some general results on fixpoints of non-monotonic operators on complete lattices the following properties of the model M(P) were proved in [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 :

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

$M_2 = \bigcap\{M : M \text{ is a supported model of } P_2 \text{ and } M \cap B_{P1} = M_1\}$,

...

$M_n = \bigcap\{M : M \text{ is a supported model of } P_n \text{ and } M \cap B_{P1 \cup ... \cup Pn-1} = M_{n-1}\}$,

v) M(P) is a model of comp(P), Clark's [C] completion of P,

vi) there is a backchaining interpreter for P using the negation as failure rule and loop checking (but working only with fully instantiated clauses) which tests for membership in M(P) when P is function-free.

This and a recent result of Lifschitz [L] showing that M(P) can be also defined using the circumscription method of McCarthy [MC] provide an ample evidence that M(P) is a natural model for a stratified program P. Other properties of M(P) were proved in Van Gelder [VG] and Przymusinski [Pr].

3. STRATIFIED PROGRAMS AND THE MAINTENANCE PROBLEM

A stratified database is a function-free stratified logic program augmented by the usual particularization axioms (see Gallaire et al. [GMN]) defining uniquely its domain and the equality predicate. P is divided into

i) a set of ground atoms defining extensional relations (Extensional Database).

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

In addition a database contains a set of integrity constraints. As the issue of integrity constraints checking for stratified databases was already studied in Lloyd and al. [LST], we do not consider them in our framework.

A stratified database P has as its intended meaning the standard model M(P). When maintaning P two representation possibilities arise :

i) implicit representation consisting just of P,

ii) explicit representation consisting of P and M(P).

Which alternative is more attractive depends on the application. For example ii) is more interesting in case of frequent queries and infrequent updates. Moreover, alternative i) leads to difficult problems concerning an efficient implementation of queries which only recently have been solved in a satisfactory way for the case of definite deductive databases (i.e those in which use of negation in the clauses is disallowed) - see Rohmer et al. [RLK] and Bancilhon et al. [BMSU]. Consequently, we choose, similarly as Nicolas and Yazdanian [NY] for the case of definite deductive databases, 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.

The maintenance problem can be viewed as a task of processing supplementary information. In the case of a stratified database P it can be formulated as follows : given P' obtained by a fact or rule insertion or deletion compute its intended meaning M(P') making use of the already existing model M(P) of P. The computation of M(P') making use of M(P) is closely related to the issue of dependency-directed backtracking discussed in [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(l), accepted(n1), ..., accepted(nk), rejected(x) <- ¬ accepted(x)}

where k,l ≥ 1 and for i = 1, ..., k $1 \leq ni \leq l$ 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, ...,l} \ {n1, ..., nk}.

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) \ {rejected(m)} U {accepted(m)}.

Similarly, a deletion of the fact accepted(nj) where $1 \leq j \leq k$ leads to a new database PODS'' with the following associated model M(PODS'')= M(PODS) \ {accepted(nj)} U {rejected(nj)}.

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.

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 the last section achieves this compromise. 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. CHOICE OF SUPPORTS

We now present various solutions to the maintenance problem. They differ in the form of supports chosen. As we analyze exclusively stratified databases, we require that, in the case of a rule insertion, the resulting program remains stratified. This can be simply checked by testing that each new arc obtained from the rule does not create in the dependency graph a cycle containing a negative arc. Also, we allow deletions only for the relations defined in the extensional part.

Let now P be a given stratified database. We assume a given maximal stratification say $P_1 \cup ... \cup P_n$ of P with the corresponding sequence of models $M_1, ..., M_n = M(P)$.

4.1 STATIC SOLUTION USING THE DEPENDENCY GRAPH

This is perhaps the simplest solution and usually the most inefficient one, but its presentation facilitates understanding of the subsequent, improved versions. 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) U 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 relation p in the model M.

LEMMA 1

i) let P' = P U $\{p(\bar{t})\}$.

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

ii) let P' = P \ $\{p(\bar{t})\}$.

If not($[r]_{M(P)} \subseteq [r]_{M(P')}$) 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) remove from M(P) all facts $r(\bar{s})$ such that p belongs to Neg(r) ; these facts all belong to M(P) \ M_{i-1} ;

2) add $p(\bar{t})$ and call the resulting set of facts M ;

3) 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($p(\bar{x})$ <- L_1 & ...& L_k)

1) add this rule to the stratum 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 (1) of the fact insertion. Call the result M ;

4) perform step (3) of the fact insertion.

FACT DELETION :

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

1) remove from M(P) all facts $r(\bar{s})$ such that p belongs to Pos(r) ;

2) remove $p(\bar{t})$ and call the resulting sets of facts M ;

3) perform step (3) of the fact insertion.

RULE DELETION :

DELETE($p(\bar{x})$ <- L_1 & ...& L_k)

1) remove this rule from the stratum 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 (1) of the fact deletion and call the resulting set of facts M ;

4) perform step (3) 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(l), late(l+1), accepted(x) <- submitted(x) & ¬rejected(x), accepted(l+1)} where $l \geq 1$.

Then M(CONF) consists of all facts already present in CONF together with the following facts : accepted(1), ...,accepted(l).

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

Thus the static analysis 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(l+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(l+1) from migration. However, this solution falls down when some trivial derivations for each fact 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, the use of negative literals complicates the issue. Each fact in the model M(P) has a new 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 rule $p(\bar{x}) \leftarrow L_1 \,\&\, \ldots \,\&\, L_k$ 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 attached to them the corresponding sets $Pos_1, \ldots, Pos_i$ and $Neg_1, \ldots, Neg_i$.

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

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

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

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 in view of the previous remark.

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. As before the Pos and Neg sets need not be disjoint.

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 from P' 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

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

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

During the updates we compute the actual form of the supports by interpreting the signed relations as follows :

$Pos' = \{q : q \text{ belongs to } Pos\} \cup Neg(r_1) \cup \ldots \cup Neg(r_j)$ where for $k = 1, \ldots, j \;\; -r_k \in Pos$,

$Neg' = \{q : q \text{ belongs to } Neg\} \cup Pos(r_1) \cup \ldots \cup Pos(r_j) \cup \{r_1, \ldots, r_j\}$ where for $k = 1, \ldots, j \;\; +r_k \in Neg$.

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

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 Neg' where Neg' is 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 Pos' where Pos' is 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 posible 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(l), accepted(x) <- submitted(x) & ¬rejected(x), accepted(l) <- submitted(l)}.

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} and Neg = {+ rejected}.

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

Pos = {submitted} and Neg = $\emptyset$.

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(l).

Though this solution leads to smaller migrations than the one given in the previous section, it can still lead to 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(l), in_program_committee(name1), ..., in_program_committee(name9), author(m1,1), ..., author(ml,l), accepted(x) <- submitted(x) & ¬ rejected(x), accepted(y) <- author(x,y) & in_program_committee(x)}

where $l \geq 1$.

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(name2,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 better but it 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 version.

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 :

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

For the situation discussed in the beginning of the previous subsection Pos and Neg sets are now updated as follows :

$Pos := Pos \cup (Pos_1 \oplus ... \oplus Pos_i) \oplus \{\{q_1, ..., q_i, \neg r_1, ..., \neg r_j\}\}$

$Neg := Neg \cup (Neg_1 \oplus ... \oplus Neg_i) \oplus \{\{+ r_1, ..., + r_i\}$ 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

$A' = \{q : q \in A\} \cup Neg(r_1) \cup ... \cup Neg(r_j)$ where for $k = 1, ..., j$ - $r_k \in A$,

and for an element A which belongs to Neg

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

i) remove from its Neg set all elements A such that p belongs to A' ;

ii) if the Neg set becomes empty remove $r(\bar{s})$ from the model.

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 acccepted(a) will have the following form :

Pos = { {submitted, rejected}, {author, in_program_committee} }

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

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 {$\emptyset$}. Since it is not empty, the fact rejected(a) is not removed from the model, as desired.

5. CLOSURE PROCESS REVIEWED

By the closure process we mean here the task of computing the model by the iterated use of the saturation. We study here the implementation of this process taking into account the additional task of constructing supports. The support is constructed in the dynamic solutions presented in the previous section refer to static information through the use of signed relation symbols. As static information can be inaccurate it is natural to seek some ways to avoid it.

5.1 CASCADE EFFECT

All solutions presented in the previous section have two phases : the removal phase during which some facts are deleted, and the addition phase during which some facts are inserted. We now present another type of solutions in which the removal and the addition phases are alternated. This will lead to improved solutions and among others will obviate the need for the static information in the supports.

We call this form of solutions "the cascade effect" because of the phenomenon produced. Consider a stratification $P = P_1 \cup ... \cup P_n$ of a stratified database P with the corresponding sequence of "layers" in M(P) : $N_1 = M_1, N_2 = M_2 \setminus M_1, ..., N_n = M_n \setminus M_{n-1}$. Now, insertions inside N_i can lead to deletions and insertions inside N_{i+1} which in turn can lead to deletions and insertions inside N_{i+2}, etc.

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.

1) THE SATURATE PROCEDURE

The purpose of this procedure is to compute the saturation using all clauses of a given stratum, and update during this computation the Pos and Neg sets of sets attached to every derived fact.

SATURATE(Stratum, B) :

Consider P_i where Stratum = P_i.

a) Compute the set $SAT(P_i,M)$ where M is the current version of the model and during this computation update each Pos and Neg sets attached to the derived facts. This time these sets are constructed as follows, assuming the situation discussed in the beginning of the subsection 4.2 :

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

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

with Pos and Neg initialised to the empty set.

b) Let B be the set of relations to which new facts were added in step (a).

2) THE REMOVEPOS PROCEDURE

The purpose of the procedure REMOVEPOS (Stratum, B, C) is to compute the set C of relations defined in the current stratum which decrease because of the decrease of those relations defined in lower strata which are listed in the set B.

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

a) remove from its Pos set all sets A such that $A \cap B \neq \emptyset$;

b) if the Pos set becomes empty, then remove $p(\bar{t})$ from M(P).
C := C U {p}.

3) THE REMOVENEG PROCEDURE

The purpose of the procedure REMOVENEG(Stratum, B, C) is to compute the set C of relations which decrease because of the increase of those relations defined in lower strata which are listed in the set B.

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

a) remove from its Neg set all sets A such that $A \cap B \neq 0$;

b) if the Neg set becomes empty, then remove $p(\bar{t})$ from M(P). C := C U {p}.

We now present the new version of the fact insertion algorithm, which uses the procedures SATURATE, REMOVEPOS and REMOVENEG.

Assume that p is defined in the stratum P_i.

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

a) add $p(\bar{t})$ with Pos = {$\emptyset$} and Neg = {$\emptyset$}.

b) Stratum := P_i ;

SATURATE(Stratum,INC);

DEC: = $\emptyset$;

WHILE Stratum $\neq P_n$

DO

i := i + 1 ;

Stratum := P_i ;

REMOVEPOS(Stratum, DEC, C_1) ;

REMOVENEG(Stratum, INC, C_2) ;

SATURATE(Stratum, B_1) ;

DEC := DEC U C_1 U C_2 ;

INC := INC U B_1

OD

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 simplify considerably the form of supports used. 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. This simplified form of supports can be efficiently implemented by simply attaching to each fact in the model the set of pointers pointing to the rules which triggered this fact during the construction of the model. Then each time during the closure process a new derivation of a fact has been found, a pointer to the last rule applied is added to the set. The actual supports in the form of Pos and Neg sets can be constructed from this set of pointers in an obvious way.

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 the set DEC U INC.

In the case of an insertion of a rule $p(\bar{x})$ <-- L_1 & ... & L_k we add it to the stratum P_i which contains the definition of the relation p and perform directly step (b) of the above algorithm. The deletions are treated in an analogous way.

To see how this version improves upon the given in subsection 4.3 one, consider the database P = {r <-- p, q <-- r, q <-- $\neg$p}. Then M(P) = {q}. INSERT(p) if 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.

5.2. SATURATION PROCESS REVIEWED

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 [C].

We exploit this independency by making use of an efficient implementation of the saturation process proposed in Rohmer et al. [RLK] for the case of definite deductive databases. This algorithm is called there the delta driven mechanism, and was firstly implemented in the framework of a relational production system [Pu].

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 hypothesis can be fired. The process stops when all increases are empty.

More formally, this algorithm has the following form :

for each relation set its increase to the initial value of the relation ;

repeat

1. determine the set H of helpful rules,

2. fire each of the rules from H once,

3. determine the increases of all relations

until no increase is registered.

Each rule is seen here as a mapping from the meanings of the relations used in its hypotheses to the meaning of the relation used in its conclusion. Here an increase of a relation is the set of its newly obtained tuples. A rule is called helpful if it uses in its hypotheses a relation whose current increase is non-empty.

The interest in the delta driven mechanism stems from the fact that it can be efficiently implemented using standard database operations, like joins and unions. However, since we also need to maintain supports attached to the facts produced, this form of implementation has to be carefully reviewed.

The supports constructed in subsections 4.2 and 4.3 use the supports already attached to individual facts derived from the hypotheses of the rule applied. To maintain these supports each newly derived fact has to be handled individually. Thus the delta driven mechanism which produces new facts in chunks cannot be applied here. On the other hand, when the form of supports proposed in the previous subsection is used, the delta driven mechanism still can be applied. Indeed, all facts produced in one delta are deduced by the same rule, so the resulting update of their supports is the same for all of them. Thus from the implementation point of view the solution proposed in this section is clearly preferable.

Note however that there is a trade-off between an efficient implementation 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 but facts 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 tc 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 should be rejected in the framework of databases. First, use of relations instead of facts in the supports allows us to use the delta driven mechanism based on relational operators to implement the closure process. Secondly, the computation costs incurred in the task of keeping all possible deductions is clearly too prohibitive to be of practical interest when many facts are present.

6. RELATED WORK

DEDUCTIVE DATABASES :

Nicolas and Yazdanian [NY] consider the maintenance problem for definite deductive databases. Absence of negation considerably simplifies the issue. 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. 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 theorical 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.

Acknowledgement The first author profitted from an early discussion on the subject of this paper with Peter van Emde Boas.

REFERENCES

[ABW] Apt, K., Blair, H. and Walker, A.,

"Towards a Theory of Declarative Knowledge", in : Proc. Workshop on Foundations of Deductive Databases and Logic Programming, Washington D.C, pp. 546-629, 1986.

[BMSU] Bancilhon, F., Maier, D., Sagiv, Y. and Ullman, J.,

"Magic Sets and Other Strange Ways to Implement Logic Programs", in : Proc. 5th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems, 1986.

[CH] Chandra, A. and Harel, D.,

"Horn Clause Queries and Generalizations", Journal of Logic Programming, vol.1, pp. 1-15, 1985.

[C] Cousot, P.,

"Asynchronous Iterative Methods For Solving a Fixed Point System of Monotone Equations in a Complete Lattice",Rapport de Recherche N 88, L.A. 7, Univ. Scientifique et Medicale de Grenoble, 1977.

[D] Doyle, J.,

"A Truth Maintenance System", Artificial Intelligence 12, pp. 231-272, 1979.

[GMN] Gallaire, H., Minker, J. and Nicolas, J-M.,

"Logic and Databases : A Deductive Approach", ACM Computing Surveys, pp. 153-185.

[VG] Van Gelder, A.,

"Negation as Failure Using Tight Derivations for General Logic Programs", in : Proc. Third IEEE Symposium on Logic Programming, Salt Lake City, Utah, 1986.

[dK] de Kleer, J.,

"An Assumption-Based Truth Maintenance System", Artificial Intelligence 28, pp. 127-162, 1986.

[L] Lifschitz, V.,

"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] Lloyd, J.W., Sonenberg, E.A. and Topor, R.,

"Integrity Constraint Checking in Stratified Databases", Technical Report 86/5, Dept. of Computer Science, Univ. of Melbourne, 1986.

[MC] McCarthy, J.,

"Circumscription - A Form of Non-monotonic Reasoning", Artificial Intelligence 13, pp. 295-323, 1980.

[MS] Martins, J.P., and Shapiro, S.C.,

"Reasoning in Multiple Belief Spaces", in : Proc. IJCAI-83, pp. 370-373, 1983.

[NY] Nicolas, J-M., and Yazdanian, K.,

"An Outline of BDGEN : a Deductive DBMS", in : Proc. IFIP-83, pp. 711-717, 1983.

[Pu] Pugin, J-M.,

"BOUM : Manuel de reference et d'utilisation", Rapport Interne du Centre de Recherche BULL, 1984.

[Pr] Przymusinski, T.,

"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.

[S1] Shepherdson, J.C,

"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] Shepherdson, J.C,

"Negation as Failure. II", Journal of Logic Programming, N 3, pp. 185-202, 1985.

[SS] Stallman, R.M. and Sussman, G.J,

"Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", Artificial Intelligence 9, pp. 135-196, 1977.

[TS] Topor, R., Sonenberg, E.A.,

"On Domain Independent Databases", in : Proc. Workshop on Foundations of Deductive Databases and Logic Programming, Washington D.C, pp. 403-419, 1986.

[STZE] Shmueli, O., Tsur, S., Zfira, H., and Ever-Hadani, R.,

"Dynamic Rule Support in Prolog", Manuscript, 1985.