On Conservative Enforced Updates *

Michael Dekhtyar[1] , Alexander Dikovsky[2] and Nicolas Spyratos[3]

[1] Dept. of CS, Tver State Univ. 3 Zheljabova str. Tver, Russia, 170000
dekhtyar@tversu.ac.ru
[2] Keldysh Institute for Applied Math. 4 Miusskaya sq. Moscow, Russia, 125047
dikovsky@spp.keldysh.ru
[3] Université de Paris-Sud, LRI, U.R.A. 410 du CNRS, Bât. 490
F-91405 Orsay Cedex, France
Nicolas.Spyratos@lri.fr

Abstract. A new method is proposed of restoring integrity constraints after committing an external update of a data base. This method is conservative in the sense that it performs the minimum of the necessary changes in the initial data base. The method is based on a novel idea of a so called preference strategy, i.e. an "oracle" which resolves globally all the conflicts, taking into account the update to be committed.

1 Introduction

There is an emerging variety of applications for which automatic enforcement of constraints is more appropriate than the traditional policy (according to which updates are rejected in case of inconsistency). Among these applications we can mention the active databases, the temporal databases and the data warehouses. In this work we propose a new approach to the automatic restoration of the integrity constraints of a database after external updates. Our approach applies to the databases with data represented by sets of literals, and integrity constraints represented by general logic programs with clauses of the form $l \ :- \ l_1, ..., l_n$, where $l, l_1, ..., l_n$ are literals. The intuitive meaning of a constraint clause is that in the case where all the literals $l_1, ..., l_n$ are in the database, the literal l must also be in the database. An external update is a pair $\Delta = (D^+, D^-)$, where D^+ is the set of literals to be added to the database and D^- is the set of literals to be removed from the database. This fits the frame proposed in [4], and like in [4] we are looking for a solution of the following problem: given an initial data base state I, a set of integrity constraints $\mathcal{P}$, and an external update Δ, a new data base state I_1 should be found, in which the update Δ is committed, and which conforms to the theory of $\mathcal{P}$. We call this problem an "enforced update problem". Various approaches to this problem have been proposed. In the early papers dealing with propositional data bases the so called revision algorithms transforming I into I_1 were described, as well as some natural criteria of

* This work was sponsored partly by INTAS (Grant 94-2412) and by the Russian Fundamental Studies Foundation (Grants 96-01-00395, 97-01-00973).

minimal change produced by the revision (cf. [8, 3]). More recent papers deal with the first order constraints, and propose solutions differing from each other in several aspects. One of the main aspects is the type of a model created by the resolving algorithm. E.g., in [9, 11] and in [12] the resulting model I_1 is stable, whereas in [1] it is well founded. The other aspect is the character of a change produced in I to receive I_1. E.g., in [9] the symmetric difference of I and I_1 is minimal with respect to the other models of constraints. The next aspect is the termination of the resolving algorithm. In [9, 11] the algorithm is partial in general, and converges under some natural restrictions. In [1] the algorithm is total. One more aspect is the type of the considered data bases. E.g., in [9, 11] and in [12] the total databases are considered, whereas in [1] the data bases are incomplete. Another aspect is the determinacy of the solution, and respectively, the confluence or the choice between several possible solutions in the case of nondeterminism. E.g., in [10] the algorithm is sequential and deterministic, in [9, 11, 12] the algorithms are nondeterministic and the resulting solutions are multiple, in [1] the algorithm is deterministic and not sequential.

We resolve the enforced update problem for incomplete data bases and with respect to the general Horn logic programs representing the integrity constraints. Unlike the other approaches our approach is concentrated first of all on the aspect of the minimality of the change produced to the initial data base. The "good model" properties of the resulting data base are regarded as second rate. We provide total nondeterministic algorithms which for any external update consistent with the constraints, deliver multiple solutions such that:

- in each solution the update is committed,
- each solution is a model of the constraints (i.e. the constraints are restored after the update),
- only minimal changes are performed in the initial data base in order to make the resulting data base satisfy the first two claims.

We call such solutions *conservative* and do not give preference of one conservative solution to the others (a real choice may depend on an implementation). The idea of the conservative solution of the enforced update problem sources from [6], where it is resolved in the case of constraints of restricted form: $l \leftarrow l_1$, (called there "literal rules"), and for updates restricted to the addition of literals to the database (the addition of an atom a is understood there as the "insertion" of a, and the addition of $\neg a$ is understood as the "deletion" of a).

The solution provided in this paper is novel from the technical point of view. It is based on the so called preference strategy, i.e. an "oracle" which resolves globally all the conflicts, taking into account the update Δ to be committed. This "combinatorial" solution hasn't any certain direction (from goal / from facts), and doesn't follow any semantical scheme (inflationary closure, stable closure, well founded closure, etc.).

The rest of the paper is organized as follows: Section 2 contains basic definitions and notations. Section 3 presents the update problem, basic results and algorithms. Section 4 discusses incrementality of the update algorithms, while Section 5 deals with their computational complexity.

2 Basic Notation and Definitions

We consider a first order signature $\mathbf{S}$ consisting of a set $\mathbf{C}$ of constants and a set $\mathbf{P}$ of predicates, and some set of variables $\mathbf{V}$. For a finite set of constants $C \subset \mathbf{C}$ in this signature $\mathbf{A}_C$ denotes the set of all atoms in $\mathbf{S}$ with variables in $\mathbf{V}$ and constants in C. $\mathbf{B}_C \subset \mathbf{A}_C$ denotes the set of all ground instances of atoms in $\mathbf{A}_C$ with respect to C. For an atom $a \in \mathbf{A}_C$ both a and $\neg a$ are *literals*. A pair of the form $(a, \neg a)$ or $(\neg a, a)$ is called a *contrary pair*. For a contrary pair (l_1, l_2) we write $l_1 = \neg.l_2$ and $l_2 = \neg.l_1$. The set of all literals over $\mathbf{V}$ and C is denoted by $\mathbf{L}_C$. For each $W \subseteq \mathbf{L}_C$ we denote by $\neg.W$ the set $\{a | \neg a \in W\} \cup \{\neg a | a \in W\}$. $\mathbf{LB}_C$ denotes the set $\mathbf{B}_C \cup \neg.\mathbf{B}_C$.
Let $\mathcal{P}$ be a finite set of clauses of the form

$$r = (l \ :- \ l_1, ..., l_n)$$

where $n \geq 0$ and l, l_i are literals in $\mathbf{L}_{C_0}$ for some $C_0 \subseteq C$. We call $\mathcal{P}$ an *update program*, or simply a *program* and the clauses of $\mathcal{P}$ *update constraints (UC)*. We denote by $ground_C(\mathcal{P})$ the set of all ground instances of clauses in $\mathcal{P}$ using constants in C. Finite subsets $I \subset \mathbf{LB}_C$ are called *DB states*. A DB state is *consistent* if it does not contain contrary pairs. A clause $r = (l \ :- \ l_1, ..., l_n)$ is *valid in a DB state* I if $l \in I$ whenever $l_i \in I$ for each $1 \leq i \leq n$. A program $\mathcal{P}$ *is valid in a DB state* I (I satisfies $\mathcal{P}$) if every clause of $\mathcal{P}$ is valid in I. This is denoted by $I \models \mathcal{P}$. Intuitively, a consistent DB-state for which $I \models \mathcal{P}$, constitutes a partial interpretation of $\mathcal{P}$. The positive literals in I are considered as true, the negative, as false, and those in $\mathbf{LB}_C \setminus I$, as unknown. A pair $\Delta = (D^+, D^-)$ where D^+, D^- are finite subsets of $\mathbf{LB}_C$, and $D^+ \cap D^- = \emptyset$, is called an *update*. Intuitively, the literals of D^+ are to be added in I, and those of D^- are to be removed from I.

We say that $\Delta = (D^+, D^-)$ is *compatible* with I if $D^+ \subseteq I$ and $D^- \cap I = \emptyset$.
An *implication tree* of a ground program $\mathcal{P}$ for a literal l is a finite tree $T(l)$ whose nodes are labeled by ground literals, the root is labeled by l and a node v is labeled by a literal l_0 if and only if there is a clause $r = (l_0 \ :- \ l_1, ..., l_n) \in \mathcal{P}$ such that if $n > 0$, then v has sons $v_1, \ldots, v_n$ labeled by $l_1, \ldots, l_n$ respectively. We denote by $cr(T)$ the *crown* of T, i.e. the set of all literals labelling the leafs of T.

3 Enforced update problem

We consider the following problem which we call an *Enforced Update Problem* (*EUP*) and which concerns active data bases [2] and deductive data bases with updates [4]. Given a DB state I, an update $\Delta = (D^+, D^-)$, and a program $\mathcal{P}$ one should find a DB state I_1 such that

- $I_1 \models \mathcal{P}$ and
- Δ is compatible with I_1.

The first claim says that one needs to restore the constraints of $\mathcal{P}$ after the

update. The second claim says that the update is enforced after the transformation of the initial DB state is committed.

We are interested in *conservative* solutions of this problem, i.e. in DB states I_1 which meet the following additional claim:

- I_1 is as "*close*" to I as possible.

This claim requires that only minimal necessary changes should be performed in the initial state in order to make the resulting DB state satisfy the first two claims.

The *EUP* is resolved in [6] for rules of the form $l \leftarrow l_1$. In [9, 11], and [1] another variant of this problem is considered in which in place of the conservative solution a solution with some "good model" property is needed. In [9] and [11] this solution should be stable in the sense of [5], whereas in [1] it should be well founded in the sense of [13]. In [9] and [11] the *EUP* may not have a solution: a $\mathcal{P}$-justified revision of I does not always exist. Our goal is to give a total algorithmic solution to the conservative *EUP* and to present some classes of programs in which our solution can have a regular control structure.

Before we proceed to the algorithms solving the conservative *EUP*, we introduce some additional notions and notation.

Definition 1 *An update* $\Delta = (D^+, D^-)$ *is* consistent *with a program* $\mathcal{P}$ *if there exists a consistent DB state* I *such that*

- $I \models \mathcal{P}$ *and*
- Δ *is compatible with* I.

In the case where Δ is consistent with $\mathcal{P}$ there evidently exists the least model of $\mathcal{P}$ compatible with Δ. We denote this model by M_Δ. $M_\Delta = lfp(T^{\in}_{\mathcal{P} \cup \Delta^+})$, therefore it is constructed in time linear with respect to the summary size of $ground(\mathcal{P})$ and Δ. Hence the consistency of $\mathcal{P}$ and Δ is recognized in the same time.

The claim of consistency of Δ with $\mathcal{P}$ is clearly a necessary premise to the *EUP*. Therefore, in the sequel we presume that this claim holds.

Let $\mathcal{P}, \Delta$, and I be given, and *con* be the set of all constants which occur in them. We consider below only the constants in *con*, i.e. only the sets $\mathbf{A}_{con}$, $\mathbf{B}_{con}$, $\mathbf{L}_{con}$, $\mathbf{LB}_{con}$, $ground_{con}(\mathcal{P})$, etc., therefore the subscript *con* will be dropped in the sequel.

Definition 2 *Let* $\mathcal{L} = (L_1, ..., L_k)$ *be a collection of finite sets of literals. A set* $H \subset \mathbf{LB}$ *is called a* hitting set *of* $\mathcal{L}$ *if* $H \cap L_i \neq \emptyset$ *for each* $1 \leq i \leq k$. H *is a minimal hitting set of* $\mathcal{L}$ *if it is a hitting set of* $\mathcal{L}$ *and no proper subset of* H *is a hitting set of* $\mathcal{L}$.

The claim of minimal changes which underlies the conservative *EUP* should be formulated in terms of some criterion of "closeness" of DB states. We propose a combination of two independent criteria: that of maximal intersection with the initial DB state, and the other, of minimal symmetric difference with respect to this DB state. We proceed from the assumption that it is more important to

keep as much initial facts as possible, than to add possibly fewer new facts. This is why we put the symmetric difference criterion on top of the intersection criterion. This is put in a precise form as follows.

Definition 3 *Let* I, I_1, I_2 *be three DB states. We say that* I_1 *is* intersection-closer *to* I *than* I_2 *(notation:* $I_1 \geq_{it}^I I_2$*) if* $I \cap I_2 \subseteq I \cap I_1$. *We write* $I_1 \equiv_{it}^I I_2$ *if* $I_1 \geq_{it}^I I_2$ *and* $I_2 \geq_{it}^I I_1$, *and we write* $I_1 >_{it}^I I_2$ *if* $I_1 \geq_{it}^I I_2$ *and* $I_2 \not\geq_{it}^I I_1$. I_1 *is* symmetric-difference-closer *to* I *than* I_2 *(notation:* $I_1 \geq_{sd}^I I_2$*) if* $I \nabla I_1 \subseteq I \nabla I_2$ *(*$I_1 \nabla I_2 = (I_1 \setminus I_2) \cup (I_2 \setminus I_1)$*). We write* $I_1 >_{sd}^I I_2$ *if* $I_1 \geq_{sd}^I I_2$ *and* $I_2 \not\geq_{sd}^I I_1$.

An EUP-algorithm is conservative *if the DB state* I_1 *which is the result of applying the algorithm to inputs* I, Δ *and* $\mathcal{P}$ *(where* Δ *is consistent with* $\mathcal{P}$*), satisfies two conditions:*

1) for no DB state $I' \models \mathcal{P}$ *compatible with an update* Δ, $I' >_{it}^I I_1$;

2) for no DB state $I' \models \mathcal{P}$ *compatible with* Δ, *and such that* $I_1 \equiv_{it}^I I'$, $I' >_{sd}^I I_1$.

The relation $I_1 \geq_{it}^I I_2$ as the concept of closeness has been used frequently, see for example [8, 3, 9, 11, 6].

The conservative nature of an *EUP* algorithm means that I_1 is *maximal* with respect to the partial order $\geq_{it}^I$, and maximal with respect to $\geq_{sd}^I$ in its $\equiv_{it}^I$*-equivalence class.*

The main idea on which our solutions of the conservative *EUP* are based is that of an "oracle" which reconciles globally all the conflicts, taking into account the update to be committed. We call this oracle a preference strategy.

Definition 4 *A* preference strategy *is any function* ρ *which maps any contrary pair* $(l, \neg.l)$ *into one of the literals* $l, \neg.l$. *A strategy* ρ *and an update* Δ *are* compatible *if for all* $l \in M_\Delta$, $\rho(l, \neg.l) = l$ *and for all* $l \in D^-$, $\rho(l, \neg.l) = \neg.l$. *A DB state* I *and a strategy* ρ *are* compatible *if for all* $l \in I$, $\rho(l, \neg.l) = l$.

We start with an *EUP* algorithm which finds a conservative solution of the *EUP* with the help of some explicit preference strategy.

***EUP*-Algorithm** $\mathcal{A}^\rho$ with a preference strategy ρ.

Input: a DB state I, an update $\Delta = (D^+, D^-)$, a program $\mathcal{P}$, and a preference strategy ρ, such that Δ is consistent with $\mathcal{P}$, and Δ is compatible with ρ.

Step 1.

Construct $ground(\mathcal{P})$.

Eliminate from $ground(\mathcal{P})$ every clause r such that for some $l \in body(r)$, $l \in D^-$ or $\neg.l \in M_\Delta$.

From the body of each remaining clause eliminate every $l \in M_\Delta$. The resulting program is denoted by $\mathcal{P}_\Delta$.

Step 2.
Construct $\tilde{I} = ((I \cup M_\Delta) \setminus \neg.M_\Delta) \setminus D^-$.
Step 3.
Construct the set $\pi = \{(l, \mathcal{L}_l) \mid l \in \mathbf{LB},\ \mathcal{L}_l = \{c \mid c \subseteq \tilde{I}$ *and* $c = cr(T(l))$ *for some implication tree* $T(l)$ *of* $\mathcal{P}_\Delta \cup \tilde{I}\}\}$.
Step 4.
$R := \{\neg.l \mid (\neg.l, \mathcal{L}_{\neg.l}) \in \pi\ \&\ \rho(l, \neg.l) = l\}$;
Step 5.
$\mathcal{L} := \bigcup\{\mathcal{L}_l \mid l \in R\}$;
Step 6.
Find some minimal hitting set H of $\mathcal{L}$.
Step 7.
$I_1 := (\tilde{I} \setminus H) \cup \{l \mid (l, \mathcal{L}_l) \in \pi$ *and* $\exists\ c \in \mathcal{L}_l (c \cap H = \emptyset)\}$.
Output: I_1.

Before we prove that the *EUP* algorithm $\mathcal{A}^\rho$ is correct we establish some of its technical properties.

Lemma 1 $M_\Delta \subseteq I_1$

Proof. First let us note that $M_\Delta \cap H = \emptyset$. Indeed, the facts $l \in M_\Delta$ do not fall into R because ρ is compatible with Δ. So they are not the leaves of the unit implication trees. Nor they are leaves of nonunit implication trees of $\mathcal{P}_\Delta \cup \tilde{I}$ because the clauses of $\mathcal{P}_\Delta$ do not contain literals of M_Δ. Let us observe that after the step 2 $M_\Delta \subseteq \tilde{I}$. Therefore, after the step 7, $M_\Delta \subseteq I_1$. □

Lemma 2 $I_1 \cap D^- = \emptyset$

Proof. Suppose that there is $d \in I_1 \cap D^-$. As $\tilde{I} \cap D^- = \emptyset$ by definition, this means that there is an implication tree $T(d)$ in $\mathcal{P}_\Delta$ with the crown $c = cr(T(d))$ such that $c \cap H = \emptyset$ and $c \subseteq \tilde{I}$. Meanwhile, $\rho(d, \neg.d) = \neg.d$ because ρ and Δ are compatible. Then after the step 4, $d \in R$ and after the step 5, $cr(T(d)) \in \mathcal{L}$. Therefore, $cr(T(d)) \cap H \neq \emptyset$ after the step 7. A contradiction. □

Lemma 3 *Let a DB state* I' *be compatible with* Δ, *and* $I' \models \mathcal{P}$. *For each implication tree* $T(l)$ *in* $\mathcal{P}_\Delta \cup I'$ *with the crown* $cr(T(l)) \subseteq I'$, *there is an implication tree* $T_1(l)$ *in* $\mathcal{P} \cup I'$ *with the crown* $cr(T_1(l)) \subseteq I'$.

Proof. (By induction on the height h of $T(l)$). If $h = 0$ then $cr(T(l)) = \{l\}$ and either $l \in I'$ or there is a fact $l \in \mathcal{P}_\Delta$. In the first case the unit implication tree $T(l)$ is in $\mathcal{P}_\Delta \cup I'$ as well. In the second case there is a clause $r = (l\ :-\ l_1, ..., l_k)$ $(k \geq 0)$ in $ground(\mathcal{P})$ which results in the fact l at the step 1. Then each literal l_i $(1 \leq i \leq k)$ belongs to M_Δ. Since I' and Δ are compatible, then $M_\Delta \subseteq I'$, and the clause r represents the tree $T_1(l)$. For $h \geq 1$ consider a clause $l\ :-\ l_1, ..., l_k\ \in \mathcal{P}_\Delta$ such that $l_1, ..., l_k$

are the sons of l in $T(l)$. By assumption each l_i has an implication tree $T_1(l_i)$ satisfying the condition of lemma. By construction of $\mathcal{P}_\Delta$ there is a clause $l :- s_1, ..., s_n \in \mathcal{P}$ such that $l_1, ..., l_k$ is a subsequence $s_{i_1}, ..., s_{i_k}$ of $s_1, ..., s_n$, and $s_j \in M_\Delta$ for each $j \notin \{i_1, ..., i_k\}$. Then $T_1(l)$ is the result of the identification of each s_{i_m} in this clause, with the root of the corresponding tree $T_1(l_m)$. This construction is correct because $M_\Delta \subseteq I'$. $\square$

Theorem 1 *Let I_1 be the DB state constructed by the algorithm $\mathcal{A}^\rho$ for a program $\mathcal{P}$, a given DB state I, an update $\Delta = (D^+, D^-)$, and a preference strategy ρ compatible with Δ. Then*

1) I_1 and ρ are compatible;

2) Δ is compatible with I_1;

3) I_1 is consistent;

4) $I_1 \models \mathcal{P}$;

5) For no DB state $I' \models \mathcal{P}$ compatible with Δ and ρ, $I' >^I_{it} I_1$;

6) For no DB state $I' \models \mathcal{P}$ compatible with Δ, and such that $I_1 \equiv^I_{it} I'$, $I' \geq^I_{sd} I_1$.

Proof. 1) Suppose that for some $l \in I_1$, $\rho(l, \neg.l) = \neg.l$. If $l \in (\tilde{I} \setminus H)$ then there is a unit implication tree $T(l)$ in $\mathcal{P}_\Delta \cup \tilde{I}$ with $cr(T(l) = \{l\}$. Then $\{l\} \subseteq \mathcal{L}_l$ and $\{l\} \in \mathcal{L}$ after the step 5. Therefore, $l \in H$ after the step 6 since l is included in any hitting set of $\mathcal{L}$, a contradiction. If, otherwise, there is such $c \in \mathcal{L}_l$ that $c \cap H = \emptyset$ then $l \in R$ after the step 4 since $\rho(l, \neg.l) = \neg.l$. Then $c \in \mathcal{L}$ after the step 5, and therefore $c \cap H \neq \emptyset$, a contradiction again.

2) This point follows immediately from lemmas 1, 2.

3) Suppose that for some l, $l \in I_1$ and $\neg.l \in I_1$. Then it follows from the definition of I_1 that l and $\neg.l$ have implication trees $T(l)$, $T(\neg.l)$ respectively in $\mathcal{P}_\Delta \cup \tilde{I}$, whose crowns do not intersect with H. Let us assume without loss of generality that $\rho(l, \neg.l) = l$. Then $\neg.l \in R$ after the step 4, and $cr(T(\neg.l)) \in \mathcal{L}$ after the step 5. Therefore after the step 6 we find that $cr(T(\neg.l)) \cap H \neq \emptyset$, a contradiction.

4) Consider some clause $r = (l :- l_1, ..., l_k) \in ground(\mathcal{P})$ $(k \geq 0)$ such that $l_i \in I_1$ for all $1 \leq i \leq k$.
Let $k = 0$. Then $l \in M_\Delta$ since $M_\Delta \models \mathcal{P}$. Hence $l \in I_1$.
If $k \geq 1$ then, as it was noticed above, there is an implication tree $T(l_i)$ in $\mathcal{P}_\Delta \cup \tilde{I}$ for each $1 \leq i \leq k$, whose crown does not intersect with H. Therefore, we can construct an implication tree $T(l)$ in $\mathcal{P}_\Delta \cup \tilde{I}$ as follows. We delete in the body of r all occurrences of literals in M_Δ and obtain a clause $l :- l_{i_1}, ..., l_{i_m} \in \mathcal{P}_\Delta$. Then we identify the resting literals $l_{i_1}, ..., l_{i_m}$ with the roots of the corresponding trees $T(l_{i_1}), ..., T(l_{i_m})$. It is clear that the crown of the so constructed implication tree $T(l)$ does not intersect with H. Thus, after the step 7, l belongs to I_1.

5) Let us consider a DB state I' compatible with Δ and ρ, such that $I' \models \mathcal{P}$ and $(I_1 \cap I) \subseteq (I' \cap I)$. Let us assume that there exists $a \in (I' \cap I) \setminus (I_1 \cap I)$. Clearly $a \notin D^-$ and $a \notin \neg.M_\Delta$, since I' is compatible with Δ. Then $a \in H$. As H is a minimal hitting set, $H' = H \setminus \{a\}$ is not a hitting set of $\mathcal{L}$. Then

there is a literal $l \in R$ and an implication tree $T(l)$ in $\mathcal{P}_\Delta \cup \tilde{I}$ such that $cr(T(l)) \cap H = \{a\}$ and $cr(T(l)) \subseteq I'$. Then by lemma 3 there exists an implication tree $T_1(l)$ in $\mathcal{P} \cup I'$ with $cr(T_1(l) \subseteq I'$. Hence, $l \in I'$ otherwise $I' \not\models \mathcal{P}$. On the other hand, $l \in R$ implies $\rho(l, \neg.l) = \neg.l$. Therefore, $l \in I'$ implies that I' is not compatible with ρ. A contradiction. So our assumption of existence of a is false, and we have $(I' \cap I) \setminus (I_1 \cap I) = \emptyset$.

6) Let $A = I \cap I_1 = I \cap I'$. Then $I \nabla I_1 = (I \setminus A) \cup (I_1 \setminus A)$ and $I \nabla I' = (I \setminus A) \cup (I' \setminus A)$. Therefore it suffices to prove that $(I_1 \setminus A) \subseteq (I' \setminus A)$. Let l be some literal in $(I_1 \setminus A)$. If $l \in M_\Delta$ then $l \in I' \setminus A$ since $M_\Delta \subseteq I'$. Assume that $l \notin M_\Delta$. Then there is an implication tree $T(l)$ in $\mathcal{P}_\Delta \cup \tilde{I}$ such that $cr(T(l)) \cap H = \emptyset$. Hence $cr(T(L)) \subseteq A \subseteq I'$. By lemma 3 there exists an implication tree $T_1(l)$ in $\mathcal{P} \cup I'$ with $cr(T_1(l)) \subseteq I'$. So finally, $l \in I' \setminus A$ since $I' \models \mathcal{P}$. $\square$

It is worth noting that the step 6 makes the algorithm $\mathcal{A}^\rho$ nondeterministic. Even for a fixed preference strategy there may exist different minimal hitting sets, and therefore, different resulting DB states I_1. Nevertheless, for each such DB state theorem 1 guarantees that it is a conservative solution of the *EUP* in our sense. Another important point to note is that theorem 1 implies the termination of the algorithm $\mathcal{A}^\rho$ on each input satisfying its conditions.

Now we proceed to a conservative *EUP*-Algorithm which is independent of specific preference strategies.

Strategy independent *EUP*-Algorithm $\mathcal{A}$.

Input: a DB state I, an update $\Delta = (D^+, D^-)$, and a program $\mathcal{P}$ which is consistent with Δ.

Step 1.
Construct $ground(\mathcal{P})$.
Construct $\mathcal{P}_\Delta$ as in the algorithm $\mathcal{A}^\rho$.
Step 2.
Construct $\tilde{I} = ((I \cup M_\Delta) \setminus \neg.M_\Delta) \setminus D^-$.
Step 3.
Construct the set $\pi = \{(l, \mathcal{L}_l) \mid l \in \mathbf{LB}, \mathcal{L}_l = \{c \mid c \subseteq \tilde{I}$ *and* $c = cr(T(l))$ *for some implication tree* $T(l)$ *of* $\mathcal{P}_\Delta \cup \tilde{I}\}\}$.
Step 4.
Guess a preference strategy ρ compatible with Δ.
$R(\rho) := \{\neg.l \mid (\neg.l, \mathcal{L}_{\neg.l}) \in \pi \ \&\ \rho(l, \neg.l) = l\}$;
Step 5.
$\mathcal{L}(\rho) := \bigcup\{\mathcal{L}_l \mid l \in R\}$;
Step 6.
Find the first minimal hitting set H of $\mathcal{L}(\rho)$ such that for no subset $H' \subset H$ and for no preference strategy ρ' compatible with Δ, H' is a hitting set of $\mathcal{L}(\rho')$.
Step 7.

$I_1 := (\tilde{I} \setminus H) \cup \{l \mid (l, \mathcal{L}_l) \in \pi \text{ and } \exists\, c \in \mathcal{L}_l (c \cap H = \emptyset)\}$.
Output: I_1.

The *EUP*-algorithm $\mathcal{A}$ is still more nondeterministic than $\mathcal{A}^\rho$. At the step 4 it guesses nondeterministically some preference strategy and then checks at the step 6 that the found ρ is optimal, i.e. there exists a hitting set H of $\mathcal{L}(\rho)$, whose proper subsets are hitting sets for no strategies compatible with Δ. This optimal H is then used at the step 7 to construct the resulting DB state I_1. Let us observe that both steps of choice: 4 and 6 are correct. Indeed, at the step 4 one really can guess some preference strategy ρ compatible with Δ, because $\mathcal{P}$ is consistent with Δ, and so $M_\Delta \cap D^- = \emptyset$. On the other hand, if there exists a minimal hitting set for some preference strategy compatible with Δ, then there evidently exists some hitting set H satisfying the condition at the step 6 (if there is a subset $H' \subset H$ which is a hitting set of $\mathcal{L}(\rho')$ for some preference strategy ρ', then one could guess ρ' in place of ρ at the step 4).

The following theorem shows that the algorithm $\mathcal{A}$ is correct and conservative.

Theorem 2 *Let I_1 be the DB state constructed by the algorithm $\mathcal{A}$ for a program $\mathcal{P}$, a given DB state I and an update $\Delta = (D^+, D^-)$. Then*
1) Δ is compatible with I_1;
2) I_1 is consistent;
3) $I_1 \models \mathcal{P}$;
4) For no DB state $I' \models \mathcal{P}$ compatible with Δ, $I' >^I_{it} I_1$;
5) For no DB state $I' \models \mathcal{P}$ compatible with Δ, and such that $I_1 \equiv^I_{it} I'$, $I' >^I_{sd} I_1$.

Proof. The only point which needs a proof different than that in the Theorem 1 is the point 4.

4) Let us assume that I' is a DB state such that $I' \models \mathcal{P}$, and I' is compatible with Δ. Let us denote $\tilde{I} \setminus I'$ by H'. Note that the assumption $I \cap I_1 \subset I \cap I'$ implies that $H' = \tilde{I} \setminus I' \subset \tilde{I} \setminus I_1 = H$. Therefore, it is sufficient to prove that H' is a hitting set for $\mathcal{L}(\rho')$ induced by some preference strategy ρ'. Let ρ' be any preference strategy such that $\rho'(l, \neg.l) = l$ if $l \in I'$. Then evidently ρ' is compatible with Δ. If we assume that H' is not a hitting set for $\mathcal{L}(\rho')$, then there exist a literal $l \in R(\rho')$ and an implication tree $T(l)$ in $\mathcal{P}_\Delta \cup I'$ such that $cr(T(l)) \cap H' = \emptyset$. Then according to lemma 3 there is an implication tree $T_1(l)$ in $\mathcal{P} \cup I'$ with $cr(T_1(l)) \subseteq I'$. Since $I' \models \mathcal{P}$, $l \in I'$ and $\rho'(l, \neg.l) = l$, whereas according to the definition of $R(\rho')$, $\rho'(l, \neg.l) = \neg.l$. A contradiction. So it is impossible that $H' \subset H$, which means that $I_1 \geq^I_{it} I'$. □

The following propositions illustrate the effect of the *EUP*-algorithm $\mathcal{A}$ in some interesting special cases.

Proposition 1 *(In notions and notation of [6]) For any consistent set of literal rules $\mathcal{P}$, any DB state I, and any update of the form $\Delta = (\{l\}, \emptyset)$, the result*

I_1 of applying the algorithm $\mathcal{A}$ is equal to the result $ins(l, I)$ of the insertion of the literal l into I.

One should simply guess any ρ compatible with Δ. For all such ρ, one and the same minimal hitting set exists (consisting of all literals l such that $\{l\} = cr(T(\neg.l_1))$ for some $l_1 \in M_\Delta$). □

Proposition 2 *Let $\Delta = (D^+, \emptyset)$ and for some DB state $J \models \mathcal{P}$, $I \cup D^+ \subseteq J$. Then the result of applying the EUP-algorithm $\mathcal{A}$ to I, Δ and $\mathcal{P}$ is a minimal DB state I_1 such that $I \cup D^+ \subseteq I_1$, and $I_1 \models \mathcal{P}$.*

One should take as a preference strategy $\rho(l, \neg.l) = l$ for $l \in J$. In this case $R(\rho) = \emptyset$ and the minimal hitting set H is also empty. □

Proposition 3 *Let $\Delta = (D^+, D^-)$ and $J \models \mathcal{P}$ for some DB state J such that $\tilde{I} \subseteq J$. Then the result I_1 of applying the EUP-algorithm $\mathcal{A}$ to I, Δ and $\mathcal{P}$ is a minimal DB state such that $\tilde{I} \subseteq I_1$ and $I_1 \models \mathcal{P}$.*

The proof is in fact the same as that of the preceding proposition. □

4 Incrementality of *EUP*-algorithms

At some steps of the *EUP*-algorithm $\mathcal{A}$ rather complex computations are performed. E.g., the construction of the minimal hitting set is a well known classical example of an NP-complete problem [7]. It would be plausible to represent these computations in a form of a converging loop on some data structure (e.g. on the list of all implication trees for some literal) such that the resulting hitting set would be the "sum" (in some sense) of the parts constructed at the consecutive steps. Unfortunately, in general this property of incrementality contradicts the property of *minimality* of a hitting set. This fact can be illustrated by the following simple example.

Example 1 *Let us consider the program $\mathcal{P}_1 = \{c :- a, b;\ \neg c :- d, f;\ g :- d, b;\ \neg g\}$, the initial DB state $\{a, b, f\}$, and the update $\Delta = (\{d\}, \emptyset)$. Let ρ be chosen as: $\rho(c, \neg c) = \neg c$, $\rho(g, \neg g) = \neg g$. Then $R(\rho) = \{c, g\}$ and $\mathcal{L}(\rho) = \{(c, \{a, b\}), (g, \{b\})\}$. If we try to construct H incrementally, and at the first step include a into H in order to cut off the crown $\{a, b\}$ of $T(c)$, then at the next step we are to include into H the literal b as well, in order to cut off the crown $\{b\}$ of $T(g)$. The so constructed set $H = \{a, b\}$ is indeed a hitting set. However it is not minimal.*

Nevertheless under certain strong though workable conditions, the incrementality can be achieved.

Definition 5 *Let* $Q \subset \mathbf{P}$ *be a finite set of predicates. A literal* $l \in \mathbf{L}$ *of the form* $p(\bar{t})$ *or* $\neg p(\bar{t})$ *is called a* Q-literal *if* $p \in Q$.

Let $k > 0$ *be an integer. We call a program* $\mathcal{P}$ k-clustered *if the set* $\mathbf{P}(\mathcal{P})$ *of all predicates used in* $\mathcal{P}$ *can be partitioned into subsets* $P_1, ..., P_m$ *such that for each* $1 \leq i \leq m$

- *the number of predicates in* P_i *does not exceed* k, *and*
- *for every clause* $l :- l_1, ..., l_r$ *in* $\mathcal{P}$, *if* l *is a* P_i*-literal, then each* l_j *is also a* P_i*-literal* $(1 \leq j \leq r)$.

The set of all clauses of $\mathcal{P}$ *which use* P_i*-literals is called a* P_i-cluster *and is denoted by* $\mathcal{P}_i$. *The partition* $\mathbf{P}(\mathcal{P}) = P_1 \cup ... \cup P_m$ *is called a* k-partition of predicates.

So a k-clustered program is partitioned into separate and independent modules-clusters, each accessing and defining a few predicates for itself. It is important to stress that the parameter k is fixed whereas the size of DB states and the number of constants available through external updates can vary. For k-clustered programs the conservative strategy independent *EUP*-Algorithm $\mathcal{A}$ can be transformed into the following incremental form.

EUP-**Algorithm** $\mathcal{A}_k^c$.

Input: a DB state I, an update $\Delta = (D^+, D^-)$ consistent with a k-clustered program $\mathcal{P}$ with a k-partition of predicates $\mathbf{P}(\mathcal{P}) = P_1 \cup ... \cup P_m$.

Step 1.
Construct $ground(\mathcal{P})$.
Construct $\mathcal{P}_\Delta$ as in the first algorithm.
Step 2.
Construct $\tilde{I} = ((I \cup M_\Delta) \setminus \neg.M_\Delta) \setminus D^-$.
FOR $i = 1$ **TO** m **DO**
Step 3.
Construct the set $\pi_i = \{(l, \mathcal{L}_l) \mid l$ *is a* $P_i - literal$, $\mathcal{L}_l = \{c \mid c \subseteq \tilde{I}$ *and* $c = cr(T(l))$ *for some implication tree* $T(l)$ *of* $\mathcal{P}_\Delta \cup \tilde{I}\}\}$.
Step 4.
Guess a preference strategy ρ over P_i, compatible with Δ.
$R_i(\rho) := \{\neg.l \mid (\neg.l, \mathcal{L}_{\neg.l}) \in \pi_i \ \& \ \rho(l, \neg.l) = l\}$;
Step 5.
$\mathcal{L}_i(\rho) := \bigcup\{\mathcal{L}_l \mid l \in R_i\}$;
Step 6.
Find the first minimal hitting set H_i of $\mathcal{L}_i(\rho)$ such that for no subset $H' \subset H_i$ and for no preference strategy ρ' over P_i compatible with Δ, H' is a hitting set of $\mathcal{L}_i(\rho')$.
END_FOR
Step 7.
$H := \bigcup_{i=1}^{m} H_i$;

$C := \bigcup_{i=1}^{m} \{l \mid (l, \mathcal{L}_l) \in \pi_i \text{ and } \exists\, c \in \mathcal{L}_l (c \cap H_i = \emptyset)\};$

$I_1 := (\tilde{I} \setminus H) \cup C.$

Output: I_1.

Theorem 3 *Let I_1 be the DB state constructed by the algorithm $\mathcal{A}_k^c$ for a k-clustered program $\mathcal{P}$, a given DB state I and an update $\Delta = (D^+, D^-)$. Then all the five assertions of theorem 2 hold for I_1.*

The proof of this theorem follows those of theorems 1, 2. One should only notice that, by the definition of k-clustering, the constructed set $H := \bigcup_{i=1}^{m} H_i$ satisfies the condition of step 6, of algorithm $\mathcal{A}$. □

5 Computational complexity

First of all, let us notice that all the described *EUP* algorithms apply only under the condition that the update Δ is consistent with $\mathcal{P}$. As it was noted above, this condition can be checked in time linear with respect to the summary size of $ground(\mathcal{P})$ and Δ.

Next we should specify for each algorithm the size of its input. For a set E of clauses or literals we denote by $|E|$ the length of the straightforward textual representation of E. A preference strategy ρ is naturally represented by a 2-dimensional table indexed by literals and containing literals. Its size will be denoted by $|\rho|$. An initial DB state I, an external update Δ, and a program $\mathcal{P}$ being given, the set of available constants con is uniquely defined, as well as the ground instance $ground_{con}(\mathcal{P})$. Therefore, one can choose as a base of the input complexity either $|\mathcal{P}|$ or $|ground_{con}(\mathcal{P})|$. The resulting complexity bounds for these two bases will differ by an exponent of the order $O(|con|^a)$ (a being the maximal arity of predicates in the input data). So we choose the former one to simplify the form of the bounds.
For $\mathcal{A}^\rho$ we take as the input size

$$N = |I| + |\Delta| + |\rho| + |ground(\mathcal{P})|.$$

For the other two algorithms the input size is

$$N = |I| + |\Delta| + |ground(\mathcal{P})|.$$

Now we proceed to the complexity of the algorithms $\mathcal{A}^\rho$, $\mathcal{A}$, and $\mathcal{A}_k^c$.

Proposition 4 *The algorithms $\mathcal{A}^\rho$, $\mathcal{A}$, and $\mathcal{A}_k^c$ can be implemented in polynomial space.*

Sketch of the proof. The matter is that :
1) all literals processed by the algorithms are in the set **lit** of literals present either in I, or in Δ, or in $ground(\mathcal{P})$, hence $M_\Delta, \tilde{I}, R$ and I_1 have the size bounded by N;

2) the steps from 3 to 6 of $\mathcal{A}^\rho$ can be implemented by the following computation, which is evidently in PSPACE:

$H := \emptyset$; $b := false$;
WHILE $\neg b$ **DO**
  FOR_ALL $l \in$ **lit DO**
    FOR_ALL $c \subseteq \tilde{I}$ **DO**
      IF $(\rho(l, \neg.l) = l)$ & $(c \cap H = \emptyset)$ &
        $(there\ exists\ T(l)\ such\ that\ c = cr(T(l)))$
      THEN $H := next(H)$
      ELSE $b := true$
      END_IF
END_FOR_ALL END_FOR_ALL END_WHILE

In this computation the function $next(H)$ enumerates all subsets of $\tilde{I}$ in the lexicographic order. The algorithms $\mathcal{A}$ and $\mathcal{A}_k^c$ are implemented in the same way if not to say about the additional loop on ρ which should be introduced. □

Remark There is another variant of the definition 5 in which k bounds the number of **literals** occurring in each cluster of $\mathcal{P}$. Let us call it a *strict k-clustering*. It applies perfectly to a narrow class of databases including the propositional databases. For strictly k-clustered programs the algorithm $\mathcal{A}_k^c$ is polynomial.

Proposition 5 *For a strictly k-clustered program $\mathcal{P}$ the algorithm $\mathcal{A}_k^c$ works in deterministic polynomial time.*

Sketch of the proof. It is easy to see that in this case for each i the sets $R_i(\rho)$ and H_i have no more than k elements, and the size of the sets π_i and $\mathcal{L}_i$ are bounded by some exponent of k. The number of different preference strategies ρ over P_i is bounded by 2^k. Therefore, each step of the loop needs time bounded by $O(2^k)$. So the total time of $\mathcal{A}_k^c$ is bounded by a polynomial of N. □

Concluding Remarks

The notions and results presented here can be extended in several ways. First of all, the notion of the validity of a clause $l \ :- \ l_1, ..., l_n$ in a database state I can be changed in several ways in order to reflect more closely the actual database practice. Indeed, the validity as defined above, requires that literal l be in I, whenever $l_1, ..., l_n$ are in I. Meanwhile, one can e.g. require that for $l = \neg a$ with a positive a $\neg a$ should not be in I whenever $l_1, ...l_n$ are in I.

Another possible extension is towards the *active rules*. An active rule is a rule of the form

ON *event* **IF** *condition* **THEN** *action*.

A constraint $l \ :- \ l_1, ..., l_n$ in our approach represents an active rule of the following restricted form:

ON $l_1, ..., l_n$ **IF** *true* **THEN** l.

Such simplified rules do not reflect relations between intended updates and committed updates (represented e.g. by the rules in [1]). We are currently looking for the possibility of adapting our algorithms to more general rules.

References

1. Bidoit, N., Maabout, S.: Update Programs Versus Revision Programs. In: *"Non-Monotonic Extensions of Logic Programming". Proc. of the Workshop at the International Logic Programming Conference, JICSLP'96.* (September 1996). (To appear in LNAI **1216**)
2. Dayal, U., Hanson,E., and Widom, J.: Active database systems. In: W. Kim, editor, *Modern Database Systems.* Addison Wesley (1995) 436-456
3. Eiter, T., Gottlob, G.: On the complexity of propositional knowledge base revision, updates, and counterfactuals. Artificial Intelligence **57** (1992) 227-270
4. Fagin, R., Kuper, G., Ullman, J., and Vardi, M.Y.: Updating Logical Databases. In: P. Kanellakis, editor, *Advances in Computing Research*, JAI Press **3** (1986) 1-18
5. Gelfond, M., Lifschitz, V.: The stable semantics for logic programs. In: R.Kovalsky and K.Bowen, editors, *Proc. of the 5th Intern. Symp. on Logic Programming.* Cambridge, MA, MIT Press (1988) 1070-1080
6. Halfeld Ferrari Alves, M., Laurent, D., Spyratos, N., Stamate, D.: Update rules and revision programs. Rapport de Recherche Université de Paris-Sud, Centre d'Orsay, LRI **1010** (12 / 1995)
7. Karp, R.M.: Reducibility among combinatorial problems. In: R.E.Miller and J.W.Thatcher, editors, *Complexity of Computer Computations.* N.-Y., Plenum Press (1972) 85-103
8. Katsuno, H., Mendelzon, A. O.: Propositional knowledge base revision and minimal change. Artificial Intelligence **52** (1991) 253-294
9. Marek, V.W., Truszciński, M.: Revision programming, database updates and integrity constraints. In: *International Conference on Data Base theory, ICDT.* LNCS **893** (1995) 368-382
10. Picouet, Ph., Vianu, V.: Expressiveness and Complexity of Active Databases. In: Afrati, F., Kolaitis, Ph., editors, *6th Int. Conf. on Database Theory, ICDT'97.* LNCS **1186** (1997) 155-172
11. Przymusinski, T.C., Turner, H.: Update by Means of Inference Rules. In: V.W.Marek, A.Nerode, M.Truszczyński, editors, *Logic Programming and Nonmonotonic Reasoning.* Proc. of the Third Int. Conf. LPNMR'95, Lexington, KY, USA (1995) 166-174
12. Raschid, L., Lobo, J.: Semantics for Update Rule Programs and Implementation in a Relational Database Management System. ACM Trans. on Database Systems **21** (December 1996) 526-571
13. Van Gelder, A., Ross, K.A., and Schlipf, J.S.: The Well-Founded Semantics for General Logic Programs. Journal of the ACM **38** (1991) 620-650