International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:9, 2007

A Logic Approach to Database Dynamic Updating

Daniel Stamate

Abstract**—** We introduce a logic-based framework for database updating under constraints. In our framework, the constraints are represented as an instantiated extended logic program. When performing an update, database consistency may be violated. We provide an approach of maintaining database consistency, and study the conditions under which the maintenance process is deterministic. We show that the complexity of the computations and decision problems presented in our framework is in each case polynomial time.

Keywords**—**Databases, knowledge bases, constraints, updates, minimal change, consistency.

I. Introduction

We introduce a logic-based framework for describing and automatically enforcing constraints on databases during updating. There is an emerging variety of applications for which automatic enforcement of constraints is more appropriate than the traditional approach (according to which updates are rejected in case of inconsistency). Among these applications we mention temporal databases and data warehouses.

In our approach, the database is seen as a set of ground literals, and each constraint is seen as a rule $l \leftarrow l_1, ..., l_n$, where $l,\ l_1, ..., l_n$ are literals. The intuitive meaning of a constraint is that, if all the literals $l_1, ..., l_n$ are in the database, then the literal l must also be in the database. Therefore the enforcement of a constraint may imply side effects, in the following sense: if the literals $l_1, ..., l_n$ are inserted in the database then it may be necessary to also insert the literal l. Such side effects are necessary in order to enforce the constraint. Constraints differ from usual Datalog rules in the following ways:

1) A constraint can have a negative literal in the head.
2) A constraint can cause side effects during updating.
3) As a consequence of 1) and 2), two constraints can be inconsistent, in the sense that their enforcement requires conflicting actions (e.g. one constraint requiring insertion of a literal l while the other requiring deletion of the *same* literal).
4) Constraints have priority over usual Datalog rules, i.e. a literal can be derived by a usual Datalog rule only if its negation is not derivable by a constraint.

Note that the set of constraints can be seen as an instantiated extended logic program (extended as rules may contain negation in their head).

One basic assumption of our approach is that both the database and the set of constraints are consistent. The database is consistent if it does not contain a literal and its negation, while a set of constraints is consistent if, starting with any literal l, it cannot generate another literal l' and its negation.

We call semantics of a database $\mathcal{L}$ with set of constraints R, the set $\mathcal{S}$ of all literals generated by $\mathcal{L}$ using the rules of R. Clearly, the semantics $\mathcal{S}$ can be inconsistent even if both $\mathcal{L}$ and R are consistent. For example, if $\mathcal{L} = \{a, \neg b\}$ and $R = \{b \leftarrow a\}$, then $\mathcal{L}$ and R are consistent, but $\mathcal{S} = \{a, b, \neg b\}$ is inconsistent.

By updating a database $\mathcal{L}$ we mean inserting or deleting a literal while leaving the database consistent. For a set of literals $\mathcal{S}$ to qualify as a possible semantics for the updated database, we impose a set of requirements similar to those of [11]. We show that, under these requirements, we can always find at least one semantics $\mathcal{S}'$ for the updated database.

In this paper, we tackle also the problem of semantic determinism of updating, i.e. we study the class of rule based constraints for which there is always exactly one possible semantics $\mathcal{S}'$ for the updated database. In this context, the main contributions of the paper are:

1) a characterization of the deterministic sets of constraints, and
2) a reduction of deterministic sets of constraints to (equivalent) sets of "simple constraints", i.e. constraints of the form $l \leftarrow l_1$.

One basic difference between our approach to updating and most other approaches, lies in the definition of the so-called minimal change requirement. Indeed, in order for a set of literals $\mathcal{S}'$ to qualify as a possible semantics for the updated database, we require the following two conditions:

1) $\mathcal{S}'$ must be as "close" as possible to the semantics $\mathcal{S}$ of the initial database
2) $\mathcal{S}'$ must contain as many as possible of the literals in $\mathcal{S}$.

Most previous approaches to updating (see for example [11]) require condition 1 but not condition 2. In these approaches "closeness" between $\mathcal{S}$ and $\mathcal{S}'$ is measured by the symmetric difference $\mathcal{S} \div \mathcal{S}'$. However, as we shall show, minimizing the symmetric difference does not necessarily mean keeping as many as possible of the literals from the initial database.

It is precisely this observation that led us to consider a different measure of change whose minimization implies that of the symmetric difference while guaranteeing, at the same time, keeping as many as possible of the literals from the initial database. Our definition of minimal change is inspired by the approach of [7].

We do not consider Datalog programs in this paper, but we can view the database itself, i.e. $\mathcal{L}$, as the result of the application of Datalog rules, on which the constraints are then enforced. That is, the idea is that the constraint rules may be viwed as being prioritary on the Datalog rules.

Several papers tackle the matter of databases/knowledge

D. Stamate is with the Department of Computing, Goldsmiths College, University of London, SE146NW London, UK (phone +44-2079197864; fax +44-2079197853; e-mail: d.stamate@doc.gold.ac.uk).

bases updating/revision using logic-based constraints. [5] extends the Revision Programming framework (a logic-based framework to express and maintain constraints on knowledge bases) with different forms of preferences. Preferences are used in order to solve the problem of nondeterminism in revising a knowledge base or logic program in presence of conflicting constraints. [2] investigates updating of knowledge bases represented by logic programs. Negative information is represented by using generalized logic programs which allow default negation also in the heads. A logic program P may be updated by another logic program U. The paper proposes a paradigm of dynamic logic programming leading to a modularisation of logic programs, in the sense mentioned above regarding the two programs. [3] presents a language capable of considering sequences of logic programs that result from the consecutive updates of an initial program, by using a priority relation among the rules of all successive programs. In [8] one provides a methodology and framework for expressing general preference information in logic programming under the answer set semantics. [1] proposes a method to compile programs formalizing update plus preference reasoning into standard generalized logic programs. [12] addresses efficiency issues during the process of integrity maintenance in deductive databases.

The rest of the paper is organized as follows: in section II we give some basic definitions and notation concerning databases. In section III, we define update semantics and we show the existence of at least one semantics for the updated database. In section IV, we focus on deterministic sets of constraints and their characterization, while in section V we show that every deterministic set of constraints can be reduced to an equivalent set of simple constraints, i.e. constraints of the form $l \leftarrow l_1$. Finally, section VI contains concluding remarks. We present also complexity results related to all computational aspects in each of the sections II, III, IV and V.

II. The Database

We view a database as a set of ground literals together with a set of constraints. In order to give the formal definition we need some notation.

We assume a fixed function-free alphabet **A** consisting of constants, variables and predicates. Given a predicate p of arity n and the terms $t_1, t_2, \cdots t_n$, the formula $p(t_1, t_2, \cdots t_n)$ is referred to as *positive literal* and the formula $\neg p(t_1, t_2, \cdots t_n)$ as a *negative literal*. A *ground* literal is a literal that contains no variables.

A constraint is a rule of the form $l \leftarrow l_1, \cdots, l_n$, where $l, l_1, \cdots, l_n$ are literals and $n \geq 1$.

Definition 2.1: - Database. A database is a set $\mathcal{L}$ of ground literals together with a set R of constraints. □

Intuitively, the literals of $\mathcal{L}$ represent real-world facts and the constraints represent properties that the facts must satisfy. Clearly, $\mathcal{L}$ changes when the real-world changes, while R may not change. For the purposes of this paper, we assume that R is fixed, so we shall say "the database $\mathcal{L}$" omitting R.

We associate every set of constants $\mathcal{C}$ with its *Herbrand base*, denoted by $HB_{\mathcal{C}}$ and consisting of all ground positive literals containing only constants from $\mathcal{C}$. For every subset S of the Herbrand base we denote by $\neg.S$ the set $\{\neg a \mid a \in S\}$. A subset D of $HB_{\mathcal{C}} \cup \neg.HB_{\mathcal{C}}$ is $consistent$ if there is no fact a such that a and $\neg a$ are both in D; otherwise D is *inconsistent*. Let $pos(D)$ and $neg(D)$ be the (unique) subsets of $HB_{\mathcal{C}}$ such that $D = pos(D) \cup \neg.neg(D)$. Obviously D is consistent if and only if $pos(D)$ and $neg(D)$ are disjoint subsets. We denote by $\neg.D$ the set $neg(D) \cup \neg.pos(D)$.

Let $\mathcal{L}$ be a database and R its set of constraints. Let $\mathcal{C}$ be a fixed set of constants containing all constants appearing in R. Let $ground(R)$ be the set of all instantiations of rules from R, in which the variables have been replaced by constants from $\mathcal{C}$. The Herbrand base of R denoted by HB_R is defined to be that of $\mathcal{C}$, i.e. $HB_R = HB_C$. Let I be a set of ground literals, i.e. let I be a subset of $HB \cup \neg.HB$. We associate R with an operator σ_R, called *semantic operator* and defined as follows:

$$\sigma_R(I) = I \cup \{l \mid l \leftarrow l_1, l_2, \cdots, l_n \in ground(R), \text{ where } l_1, l_2, \cdots l_n \in I\}.$$

When the set of constraints is understood from context, we simply write σ instead of σ_R. It is obvious that σ_R is a monotone operator, with respect to set inclusion, so the sequence:

$$\sigma^0(I) = I \quad \text{and} \quad \sigma^n(I) = \sigma(\sigma^{(n-1)}(I)), \text{ for every integer } n > 0,$$

has a limit. This limit, denoted by $lfp(\sigma, I)$ or by σ_I is referred to as the *least fixpoint of σ with respect to I*. If l is a ground literal, we denote $\sigma_{\{l\}}$ by σ_l. As σ is monotone, we can prove that $lfp(\sigma, I)$ is also monotone in its second argument (with respect to set inclusion).

Proposition 2.1: If I and J are two sets of ground literals such that $I \subseteq J$ then $\sigma_I \subseteq \sigma_J$, i.e. $lfp(\sigma, I)$ is monotone in its second argument. □

The following lemma characterizes a set I of ground literals deductively closed under the rules of R, i.e. a set I such that $\sigma(I) \subseteq I$.

Lemma 2.1: $\sigma(I) \subseteq I$ iff $\sigma_I = I$. □

Note that σ_I may be an inconsistent set, and that inconsistency of σ_I may be caused either by the constraints of R or by the literals of I. For example, if $I = \{a\}$ and $R = \{b \leftarrow a, \neg b \leftarrow a\}$ then we have: $\sigma_I = \{a, b, \neg b\}$, which is an inconsistent set.

Definition 2.2: - Database Semantics. The semantics of a database $\mathcal{L}$ is defined to be the least fixpoint of σ with respect to $\mathcal{L}$, namely $\sigma_{\mathcal{L}}$. A database $\mathcal{L}$ is called consistent if its semantics is consistent. □

It is not difficult to show that the semantics of a database is computed in polynomial time w.r.t. the size of the set of stored facts.

Definition 2.3: - Consistent constraints. We say that R is a consistent set of constraints if for any ground literal l there exists a set of ground literals S such that $\sigma_{S \cup \{l\}}$ is a consistent set. □

In other words, R is consistent if for any ground literal l there exists a consistent database containing l. We have the following characterization of constraint consistency:

Proposition 2.2: R is consistent if and only if σ_l is a consistent set for any ground literal l. □

Example 2.1: Let R be $\{a \leftarrow b, c;\ \ \neg a \leftarrow b, c\}$. Obviously R is a consistent set of constraints since for any ground literal l, $\sigma_l = \{l\}$ is consistent. Note that R expresses the fact that b and c cannot be both in the database. □

From now on we shall deal only with consistent sets of constraints. That is, when talking about sets of constraints, even if not mentioned, they will be supposed to be consistent. Moreover, we assume that all constraints are instantiated. So from now on, $\mathcal{L}$ is a set of ground literals and R is a set of instantiated rules.

III. Update Semantics

Let $\mathcal{L}$ be a consistent database with constraints R and semantics $\mathcal{S}$ (i.e. $\mathcal{S} = \sigma_{\mathcal{L}}$). Updating $\mathcal{L}$ means inserting or removing a ground literal l, while leaving the database consistent. Inserting l means that l must be in the semantics of the updated database, while removing l means that l must not be in the semantics of the updated database. We denote by $in(l)$ the insertion of l and by $out(l)$ the deletion of l.

In order for an update to be acceptable, the semantics $\mathcal{S}'$ of the updated database must satisfy certain conditions. Before giving formal definitions, we describe these conditions informally, as follows:

1) $\mathcal{S}'$ must contain l in the case of insertion, and must not contain l in the case of deletion
2) $\mathcal{S}'$ must be consistent
3) $\mathcal{S}'$ must be deductively closed under the rules of R
4) Every literal of $\mathcal{S}'$ must be "justifiable" from those literals of $\mathcal{L}$ that are still present in $\mathcal{S}'$ (and from the inserted literal l, in the case of insertion).
5) $\mathcal{S}'$ must represent a "minimal change" with respect to the semantics $\mathcal{S}$ of the initial database.

Any set of literals $\mathcal{S}'$ that satisfies the above conditions is an acceptable semantics for the updated database. In general, there are 0, 1 or more sets $\mathcal{S}'$ that satisfy these conditions. In this paper, we are interested in sets R of constraints that admit exactly one set $\mathcal{S}'$, for any insertion or deletion of a literal.

Although conditions 1 to 3 above are self-explanatory, conditions 4 and 5 need further explanation.

Condition 4 simply says that for a literal to be included in $\mathcal{S}'$, it must be derivable (using the rules of R) from already "known" literals. Known literals are those literals of $\mathcal{L}$ that remain in the database after updating, and the literal l being inserted (in the case of insertion). For example, consider the database:

$$\mathcal{L} = \{a, b\}, \qquad R = \{c \leftarrow a, b;\ d \leftarrow \neg c, b\}$$

and suppose that we want to insert $\neg c$. Here $\mathcal{S}'$ can be $\{\neg c, b, d\}$. Indeed, every literal from $\mathcal{S}'$ can be derived from $(\mathcal{S}' \cap \mathcal{L}) \cup \{\neg c\} = \{b, \neg c\}$, where b is the literal of the initial database that still belongs to the updated database, and $\neg c$ is the inserted literal. For more details see [11] which first introduced the notion of "justification".

Concerning Condition 5, the change when going from $\mathcal{S}$ to $\mathcal{S}'$ can be measured using the symmetric difference: $\mathcal{S} \div \mathcal{S}' = (\mathcal{S} \setminus \mathcal{S}') \cup (\mathcal{S}' \setminus \mathcal{S})$. Indeed, $\mathcal{S} \setminus \mathcal{S}'$ is the set of literals that are removed from $\mathcal{S}$ during updating (i.e. when going from $\mathcal{S}$ to $\mathcal{S}'$), while $\mathcal{S}' \setminus \mathcal{S}$ is the set of literals that are added to $\mathcal{S}$. Thus $\mathcal{S} \div \mathcal{S}'$ represents the total change during updating. It follows that, to satisfy Condition 5, it suffices to require that $\mathcal{S} \div \mathcal{S}'$ be minimal (with respect to set inclusion), and this is the approach taken by several authors [11] [4]. Similarly, in [6] one requires that $|\mathcal{S} \div \mathcal{S}'|$ be minimal.

In this paper, we provide a measure of change which is finer than symmetric difference, in the sense that minimal change with respect to our measure implies minimality of the symmetric difference. Roughly speaking, among the sets $\mathcal{S}'$ that satisfy conditions 1 to 4, and minimize $\mathcal{S} \div \mathcal{S}'$, our measure chooses one that minimizes the difference $\mathcal{S} \setminus \mathcal{S}'$, i.e. one that removes as few as possible among the literals of $\mathcal{S}$ (or equivalently, one that keeps as many as possible of the literals in $\mathcal{S}$). Let us see an example.

Example 3.1: Consider the database :
$\mathcal{L} = \{b, c\}$, $R = \{d_1 \leftarrow a, c;\ d_2 \leftarrow a, c;\ d_3 \leftarrow a, c\}$
whose semantics is $\mathcal{S} = \sigma_{\mathcal{L}} = \{b, c\}$, and assume that we want to insert a. Two possible semantics for the updated database are: $\mathcal{S}_1 = \{a, b, c, d_1, d_2, d_3\}$ and $\mathcal{S}_2 = \{a, b\}$. Intuitively, $\mathcal{S}_1$ is obtained by just adding a to the database *without* removing any literal of $\mathcal{S}$, whereas $\mathcal{S}_2$ is obtained by adding a, but this time also removing c from $\mathcal{S}$. Both $\mathcal{S}_1$ and $\mathcal{S}_2$ satisfy conditions 1 to 4, and moreover both minimize the symmetric difference. However, of these two possible semantics, we choose $\mathcal{S}_1$ because it preserves more literals of $\mathcal{S}$ than $\mathcal{S}_2$ does. □

We would like to emphasize that minimal change based only on minimization of the symmetric difference does not guarantee that as much as possible of the initial semantics is preserved. Indeed, in the example above $\mathcal{S}_2$ does not preserve the maximum from the semantics of the initial database, although it minimizes the symmetric difference.

In view of our previous discussion, we feel justified in defining minimal change by giving priority to the minimization of $\mathcal{S} \setminus \mathcal{S}'$ over minimization of the symmetric difference $\mathcal{S} \div \mathcal{S}'$. We do this by:

1) defining our measure of change to be the pair $(\mathcal{S} \setminus \mathcal{S}', \mathcal{S}' \setminus \mathcal{S})$, denoted by $\mathcal{S} \bigtriangledown \mathcal{S}'$, and
2) defining a partial order $\preceq$ between pairs of sets of literals as follows: $(A_1, A_2) \preceq (B_1, B_2)$ iff $A_1 \subset B_1$ or $(A_1 = B_1\ and\ A_2 \subseteq B_2)$.

Obviously, this partial order gives priority to minimality of the first component of a pair, as required. We have the following result.

Proposition 3.1: Let T be a family of sets of ground literals, and S a fixed set of ground literals. Let S_1 be a set of ground literals from T such that $S \bigtriangledown S_1$ is minimal w.r.t. $\preceq$ among all $S \bigtriangledown X$ with $X \in T$. Then
(1) $S \setminus S_1$ is minimal w.r.t. set inclusion among all $S \setminus X$ with $X \in T$
(2) $S \div S_1$ is minimal w.r.t. set inclusion among all $S \div X$ with $X \in T$. □

Suppose now that the family T in the above proposition is the family of all candidate semantics X for the updated database. Then what Proposition 3.1 says is that minimality

of $S \bigtriangledown X$ with respect to $\preceq$ implies minimality of both, $S \setminus X$ and $S \div X$, with respect to set inclusion.

Given a database $\mathcal{L}$ with constraints R, and two sets of ground literals X and Y, we say that:

1) X satisfies a rule $r = l \leftarrow l_1, ..., l_n$, if $l_1, ..., l_n \in X$ implies $l \in X$.
2) X is a model of R, denoted $X \models R$, if X is consistent and X satisfies every rule in R.
3) X generates the literal l in R, denoted $X \vdash_R l$ (or $X \vdash l$ if R is understood) if $l \in \sigma_X$.
4) X generates Y in R, denoted $X \vdash_R Y$ (or $X \vdash Y$ if R is understood), if $X \vdash_R l \quad \forall l \in Y$.

We are ready now to state formally the requirements that the semantics of an updated database should satisfy.

Definition 3.1: - in-semantics. Let l be a ground literal, and let $\Delta = (\mathcal{L}, R)$ be a consistent database with semantics $\mathcal{S}$. A set of ground literals $\mathcal{S}'$ is called in-semantics for l and $\mathcal{L}$ if the following requirements (called in-requirements) hold:

1) $l \in \mathcal{S}'$
2) $\mathcal{S}'$ is consistent
3) $\mathcal{S}'$ satisfies the constraints of R
4) $(\mathcal{S}' \cap \mathcal{L}) \cup \{l\} \vdash \mathcal{S}'$
5) under the requirements 1 to 4, $\mathcal{S} \bigtriangledown \mathcal{S}'$ is minimal w.r.t. $\preceq$. □

The in-requirements 1-3 simply say that $\mathcal{S}'$ is a model of R which contains l. The in-requirement 4 expresses the justification condition for the elements of the updated semantics, as discused earlier. The in-requirement 5 expresses minimality of change, also discussed earlier.

Definition 3.2: - out-semantics. Let l be a ground literal, and let $\Delta = (\mathcal{L}, R)$ be a consistent database with semantics $\mathcal{S}$. A set of ground literals $\mathcal{S}'$ is called out-semantics for l and $\mathcal{L}$ if the following requirements (called out-requirements) hold:

1) $l \notin \mathcal{S}'$
2) $\mathcal{S}'$ is consistent
3) $\mathcal{S}'$ satisfies the constraints from R
4) $(\mathcal{S}' \cap \mathcal{L}) \vdash \mathcal{S}'$
5) under requirements given above, $\mathcal{S} \bigtriangledown \mathcal{S}'$ is minimal w.r.t. $\preceq$. □

We denote by *in-sem*$(l, \mathcal{L}, R)$ and *out-sem*$(l, \mathcal{L}, R)$ the family of in-semantics and the family of out-semantics respectively. We shall prove that these families are nonempty, that is, there always exists at least one in(out)-semantics, for any ground literal l. In other words, any insertion or deletion is possible to perform in a consistent database, and the result of the update (even if non deterministic) is a consistent database.

First, let us give an equivalent characterization of in(out)-semantics, using the set of literals that are still present in the semantics $\mathcal{S}'$ of the updated database.

Proposition 3.2: Let $\mathcal{L}$ be a database with semantics $\mathcal{S}$. Let l be a ground literal, $\mathcal{S}'$ a set of ground literals satisfying the in(out)-requirements 1-4 and $B = \mathcal{S}' \cap \mathcal{L}$. Then the following statement holds:

$\mathcal{S} \bigtriangledown \mathcal{S}'$ is minimal (w.r.t. $\preceq$) iff B is maximal (w.r.t. set inclusion). □

Note that the above proposition provides a (equivalent) reformulation of in(out)-requirement 5. We shall refer to this reformulation as in(out)-requirement 5', i.e. we have:

$B = \mathcal{S}' \cap \mathcal{L}$ is maximal (w.r.t. set inclusion).

We need the following lemma:

Lemma 3.1: Let $\mathcal{L}$ be a database with semantics $\mathcal{S}$. If l is a ground literal, $\mathcal{S}'_1$ and $\mathcal{S}'_2$ are sets of ground literals satisfying in(out)-requirements 1-4 and $B_1 = \mathcal{S}'_1 \cap \mathcal{L}$ and $B_2 = \mathcal{S}'_2 \cap \mathcal{L}$ then the following statements hold:

1) $\mathcal{S}'_1 = \sigma_{B_1 \cup \{l\}}$ (in the case of insertion) and $\mathcal{S}'_1 = \sigma_{B_1}$ (in the case of deletion)
2) $\mathcal{S} \bigtriangledown \mathcal{S}'_1 \preceq \mathcal{S} \bigtriangledown \mathcal{S}'_2$ iff $B_1 \supseteq B_2$
3) $\mathcal{S} \bigtriangledown \mathcal{S}'_1 = \mathcal{S} \bigtriangledown \mathcal{S}'_2$ iff $B_1 = B_2$. □

The following two propositions state how the families of in- and out-semantics can be computed.

Proposition 3.3: Let $\Delta = (\mathcal{L}, R)$ be a consistent database and l be a ground literal. Then we have *in-sem*$(l, \mathcal{L}, R) = \{\sigma_{B \cup \{l\}} \mid B$ maximal with $B \subseteq \mathcal{L}$, and $\sigma_{B \cup \{l\}}$ consistent$\}$. □

Proposition 3.4: Let $\Delta = (\mathcal{L}, R)$ be a consistent database and l be a ground literal. Then we have *out-sem*$(l, \mathcal{L}, R) = \{\sigma_B \mid B$ maximal with $B \subseteq \mathcal{L}$ and $l \notin \sigma_B\}$. □

Through the following results we justify the fact that in a consistent database any update is accepted and can be performed, i.e. there exist at least one in(out)-semantics and a database with this semantics.

Proposition 3.5: For any consistent database $\Delta = (\mathcal{L}, R)$ and for any update $in(l)$ or $out(l)$ there exists at least one in-semantics and at least one out-semantics, respectively. □

One interesting property of our update semantics is the so called "monotonicity", in the following sense: suppose that a database $\mathcal{L}_1$ is included in a database $\mathcal{L}_2$ (over the same set of constraints), and thus that the semantics $\mathcal{S}_1$ of $\mathcal{L}_1$ is included in the semantics $\mathcal{S}_2$ of $\mathcal{L}_2$. Suppose now that we insert or delete the same literal in $\mathcal{L}_1$ and in $\mathcal{L}_2$, and let $\mathcal{S}'_1$ be some semantics of the updated databases obtained from $\mathcal{L}_1$. Then there exists a semantics $\mathcal{S}'_2$ of the updated database obtained from $\mathcal{L}_2$, such that $\mathcal{S}'_1$ is included in $\mathcal{S}'_2$.

Theorem 3.1: Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two consistent databases over the same set of constraints R, such that $\mathcal{L}_1 \subseteq \mathcal{L}_2$, and let l be a ground literal. Then for any semantics $\mathcal{S}'_1 \in$ *in(out)-sem*$(l, \mathcal{L}_1, R)$ there exists a semantics $\mathcal{S}'_2 \in$ *in(out)-sem*$(l, \mathcal{L}_2, R)$ such that $\mathcal{S}'_1 \subseteq \mathcal{S}'_2$. □

We conclude this section by a complexity result expressing that, given any update, the set of all update semantics can be computed in polynomial time w.r.t. the size of the set of stored facts.

Proposition 3.6: If l is a literal and $\Delta = (\mathcal{L}, R)$ is a database then the set of all in(out)-semantics for l and Δ is computed in polynomial time w.r.t. the size of $\mathcal{L}$. □

IV. Deterministic Sets of Constraints

In the previous section, we saw a method for inserting or deleting a literal l in a database consisting of a set of literals $\mathcal{L}$ and a (fixed) set of constraints R. Our method requires

that the semantics $\mathcal{S}'$ of the updated database satisfies certain conditions, called "in-requirements" in the case of insertion (see Definition 3.1) and "out-requirements" in the case of deletion (see Definition 3.2).

We have seen that, under the assumption that the initial database is consistent, there is at least one semantics $\mathcal{S}'$ satisfying the in-requirements, and at least one semantics $\mathcal{S}''$ satisfying the out-requirements (see Proposition 3.5). Therefore, the insertion or the deletion of a literal is always possible, *but* we may have non-determinism, i.e. we may have to choose among many possible semantics $\mathcal{S}'$ for the updated database. Let us see an example.

Example 4.1: Let $\mathcal{L} = \{b, c\}$ and $R = \{a \leftarrow b, c\}$. Obviously, the semantics of this database is $\mathcal{S} = \{a, b, c\}$. Suppose now that we want to insert the literal $\neg a$. Then we can show that the sets $\mathcal{S'}_1 = \{\neg a, b\}$ and $\mathcal{S'}_2 = \{\neg a, c\}$ both qualify as in-semantics, as they both satisfy the in-requirements.

Intuitively $\mathcal{S'}_1$ corresponds to adding $\neg a$ to $\mathcal{L}$ and removing b (in order to preserve the consistency), and $\mathcal{S'}_2$ corresponds to adding $\neg a$ to $\mathcal{L}$ and removing c. By the way, $\mathcal{S'}_1$ and $\mathcal{S'}_2$ are the only possible in-semantics, since there is no other maximal subset B_3 of $\mathcal{L} = \{b, c\}$ different than $B_1 = \mathcal{S'}_1 \cap \mathcal{L} = \{b\}$ and $B_2 = \mathcal{S'}_2 \cap \mathcal{L} = \{c\}$ and such that $\sigma_{B_3 \cup \{\neg a\}}$ is consistent (see Proposition 3.3). □

In this section, we address the following problem: characterize the sets of constraints R for which there is one and only one semantics $\mathcal{S}'$ for the updated database. We refer to such sets of constraints R as *deterministic* sets of constraints.

Definition 4.1: A set of constraints R is called in(out)-deterministic if for any consistent database $(\mathcal{L}, R)$ and for any ground literal a, there exists a unique in(out)-semantics. R is called simply deterministic if it is both in-deterministic and out-deterministic. □

In order to characterize determinism, we associate each set of constraints R with the following set:
$Inc_R = \{M \mid M$ minimal with $M \subseteq HB \cup \neg.HB, \exists c \in HB \cup \neg.HB$ s.t. $M \vdash_R \{c, \neg c\}\}$

Intuitively, each set M in Inc_R is a minimal set of literals that can generate inconsistency, using the rules of R. Therefore, any superset of a set in Inc_R has that same property. We have the following result:

Theorem 4.1: R is in-deterministic iff every set in Inc_R has exactly two elements. □

We characterize now the sets of constraints that are out-deterministic. To this end, given any ground literal l, we define the following set:
$Premise_R(l) = \{M \mid M$ minimal with $M \subseteq HB \cup \neg.HB, M \vdash_R l, \forall c \in HB \cup \neg.HB, M \nvdash_R \{c, \neg c\}\}$

Intuitively, each set M in $Premise_R(l)$ is a minimal set of literals that can generate l and that does not generate inconsistency, using the rules of R.

Theorem 4.2: R is out-deterministic iff every set from $Premise_R(l)$ is a singleton, for every ground literal l. □

We conclude this section by a complexity result regarding the decision problem of deterministic set of constraints.

Lemma 4.1: For any set of constraints R and literal l the sets $Premise_R(l)$ and Inc_R can be computed in polynomial time w.r.t. the size of R. □

Proposition 4.1: Let R be a set of constraints. The problem whether R is deterministic can be decided in polynomial time w.r.t. the size of R. □

V. The Reduction of Deterministic Sets of Constraints

In the previous section, we have characterized sets of constraints with an important semantic property, namely, sets of constraints for which there is only one semantics for the updated database. We have called such sets deterministic.

In this section, we show that the deterministic sets of constraints have also an important syntactic property, namely, they can be reduced to sets of simpler constraints containing exactly one literal in their body.

Definition 5.1: We call *simple constraint* any constraint of the form $l_0 \leftarrow l$, i.e. any constraint with exactly one literal in the body. □

In what follows, we first show that every set of simple constraints is deterministic, and then we show a sort of converse, namely, that every deterministic set of constraints can be "reduced" to a set of simple constraints (modulo an extension of the alphabet).

Theorem 5.1: Any set of simple constraints is deterministic. □

To simplify terminology, we shall say "deterministic set" instead of "deterministic set of constraints" and "simple set" instead of "set of simple constraints". We have just seen that every simple set is a deterministic set (Theorem 5.1), but obviously not every deterministic set is a simple set. However, as we shall see, every deterministic set can be reduced to a simple set with the "same behaviour". First, let us see an example.

Example 5.1: Consider the database $\mathcal{L} = \{b, c, d\}$ with the set of constraints:

$$R = \{a \leftarrow b, c, d; \quad b \leftarrow c; \quad c \leftarrow d\}$$

where R is not a simple set, and suppose that we want to delete a. At first glance, this deletion looks non-deterministic, as there are three possible solutions: (1) remove b, (2) remove c, or (3) remove d.

But let us observe that these solutions have the following side effects: (1) the first solution causes the removal of c and d (due to the second and third rule), (2) the second causes the removal of d (due to the third rule), and (3) the third solution has no side effects.

As we want to modify minimally the database, only the third solution is acceptable, since it supposes only removing d, that anyway is required additionally by the first two solutions. Intuitively, although R is not a simple set, it behaves exactly as the following simple set:

$$R' = \{a \leftarrow d; \qquad b \leftarrow c; \qquad c \leftarrow d\}$$

So we expect R to be deterministic. Indeed, in order to check this, we apply Theorems 4.1 4.2. One can prove that
$Inc_R = \{\{d, \neg a\}, \{d, \neg b\}, \{d, \neg c\}, \{c, \neg b\}\} \cup \{\{l, \neg l\} \mid \forall l \in$

$HB\}$ and
$Premise_R(a) = \{\{a\}, \{d\}\}$,
$Premise_R(b) = \{\{b\}, \{c\}, \{d\}\}$,
$Premise_R(c) = \{\{c\}, \{d\}\}$,
$Premise_R(l) = \{\{l\}\}$, for every ground literal $l \notin \{a, b, c\}$. □

In order to formalize the notion of "same behaviour" from the previous example, we need some preliminary definitions and notation.

Let R_1 and R_2 be two sets of constraints. We say that R_1 implies R_2, denoted by $R_1 \models R_2$, if for any consistent set of ground literals X, $X \models R_1$ implies $X \models R_2$. We say that R_1 and R_2 are equivalent, denoted by $R_1 \equiv R_2$, if $R_1 \models R_2$ and $R_2 \models R_1$. We denote by $SAT(R_1)$ the set of all models of R_1. Clearly, $R_1 \models R_2$ iff $SAT(R_1) \subseteq SAT(R_2)$, and $R_1 \equiv R_2$ iff $SAT(R_1) = SAT(R_2)$

For instance, in the previous example, the deterministic set R is equivalent to the simple set R', i.e. $R \equiv R'$. This is not true in general, as one can show that not every deterministic set is equivalent to a simple set.

However, the idea here is to capture the behaviour of a non-simple set using a simple one. That is, we reduce a set of constraints to another set of a specific type (here, to a simple set), whose behaviour we know.

From now on, every set of constraints R will be considered together with a set of predicate symbols $\mathcal{A}_R$ such that all literals appearing in R are constructed with predicates from $\mathcal{A}_R$. Additionally, all updates on any database with constraints R concern literals constructed with predicates from $\mathcal{A}_R$. If M is a set of ground literals, we define the *projection* of M on a set of predicates A, denoted by M/A, to be the set of all literals from M constructed with predicates from A. We extend the projection operation to a family of sets $\mathcal{W}$ in a natural way, namely $\mathcal{W}/A = \{M/A \mid M \in \mathcal{W}\}$. Let us now define formally the reduction relation.

Definition 5.2: Let R_1 and R_2 be two sets of constraints with semantic operators σ^1 and σ^2, and sets of predicate symbols A_1 and A_2, respectively, such that $A_1 \subseteq A_2$. We say that R_1 reduces to R_2, denoted by $R_1 \triangleright R_2$, if $SAT(R_1) = SAT(R_2)/A_1$. □

We have the following result:

Theorem 5.2: If $R_1 \triangleright R_2$ then for every set of ground literals $\mathcal{L}$ the following statements hold:

1) $\sigma^1_{\mathcal{L}}$ is consistent iff $\sigma^2_{\mathcal{L}}$ is consistent
2) if $\sigma^1_{\mathcal{L}}$ and $\sigma^2_{\mathcal{L}}$ are consistent, then $\sigma^1_{\mathcal{L}} = \sigma^2_{\mathcal{L}}/A_1$. □

What the above theorem says is that a database $\mathcal{L}$ is consistent w.r.t. R_1 iff it is consistent w.r.t. R_2, and that the semantics w.r.t. R_1 can be obtained by projection on A_1 of the semantics w.r.t. R_2.

Let us see now how the two sets of constraints behave w.r.t. updates.

Theorem 5.3: Let R_1, R_2 be two sets of constraints such that $R_1 \triangleright R_2$. Then for every set of ground literals $\mathcal{L}$ such that $(\mathcal{L}, R_1)$ is consistent, and for every ground literal l constructed with predicates from A_1, the following statements hold:

1) *in-sem*$(l, \mathcal{L}, R_1) =$ *in-sem*$(l, \mathcal{L}, R_2)/A_1$
2) *out-sem*$(l, \mathcal{L}, R_1) =$ *out-sem*$(l, \mathcal{L}, R_2)/A_1$. □

The above theorem states in fact that the two sets of constraints have the same behaviour w.r.t. updates.

Theorem 5.4: For any deterministic set R_1 there exists a simple set R_2 computed in polynomial time w.r.t. the size of R_1 such that $R_1 \triangleright R_2$. □

Theorems 5.2, 5.3 and 5.4 justify our claim that the behaviour of deterministic sets is entirely captured by simple sets, that in turn form a proper subclass of the class of deterministic sets.

VI. Concluding Remarks

We have introduced a logic-based framework for database updating under constraints. We have defined the database semantics and updated database semantics, and we have tackled the problem of nondeterminism during updating. This nondeterminism appears while attempting to restore the database integrity w.r.t. the set of constraints. We have focused on *semantic determinism*, i.e. on the characterization of sets of constraints for which there is (always) exactly one possible semantics for the updated database. We have also seen that every deterministic set of constraints can be reduced to a set of simple constraints (which are easier to manage, both conceptually and computationally) with the same behaviour w.r.t. updates.

We have shown also that the complexity of computational aspects and decision problems presented in our framework is in each case polynomial time.

References

[1] J. J. Alferes, P. Dell'Acqua, L. M. Pereira., *A compilation of updates plus preferences*, Technical report, LiTH-ITN-R-2002-7, Dept. of Science and Technology, Linkoping University, Sweden, 2002
[2] J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, T. C. Przymusinski, *Dynamic updates of non-monotonic knowledge base*, J. of Logic Programming, 45:1-3, 2000
[3] J. J. Alferes, L. M. Pereira, *Updates plus Preferences*, Procs. of JELIA'00, 2000.
[4] A. Borgida, *Language features for flexible handling of exceptions in information systems.*, J. ACM Trans. Database Syst. 10, 1985
[5] G. Brewka, T. Eiter, *Preferred Answer Sets for Extended Logic Programs*, Proc. of KR'98: Principles of Knowledge Representation and Reasoning, 1998
[6] M. Dalal, *Updates in propositional databases.* Tech. Report DCS-TR-222, Dept. of Comp. Science, Rutgers University, New Brunswick, 1988
[7] M. Dekhtyar, A. Dikovsky, N. Spyratos, *On Conservative Enforced Updates.* Proc. of 4^{th} Int. Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'97, 1997
[8] J. Delgrande, T. Schaub, H. Tompits, *A Framework for Compiling Preferences in Logic Programs* J. Theory and Practice of Logic Programming, 2002
[9] M. Halfeld Ferrari Alves, D. Laurent, N. Spyratos, *Update Rules in Datalog programs.* Proc. of 2^{nd} Int. Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'95, 1995
[10] H. Katsuno, A. Mendelzon *Propositional Knowledge Base Revision and Minimal Change.* J. Artifi cial Intelligence, 1991:52
[11] V.W. Marek, M. Truszcyński, *Revision Programming, Database Updates and Integrity Constraints.* Proc. of International Conference on Database Theory, ICDT, 1995
[12] E. Mayol, E. Teniente, *Structuring the Process of Integrity Maintenance*, Proc. of 8th Database and Expert Systems Applications - DEXA'97, 1997