Extended Deductive Databases with Uncertain Information

Daniel Stamate

Abstract—The paper presents an approach for handling uncertain information in deductive databases using multivalued logics. Uncertainty means that database facts may be assigned logical values other than the conventional ones - true and false. The logical values represent various degrees of truth, which may be combined and propagated by applying the database rules. A corresponding multivalued database semantics is defined. We show that it extends successful conventional semantics as the well-founded semantics, and has a polynomial time data complexity.

Keywords—Reasoning under uncertainty, multivalued logics, deductive databases, logic programs, multivalued semantics.

I. INTRODUCTION

REAL world information is not always exact, but mostly imperfect, in the sense that we handle estimated values, probabilistic measures, degrees of uncertainty, etc., rather than exact values and information which is certainly true or false.

In the conventional database systems the information handled is assumed to be rather exact, as a tuple (or fact) either is in the database (so it is true) or is not (so it is false). That is, in such a context we use quite simplified models of the "real" world.

In this paper we present an approach for querying and updating extended deductive databases with uncertain/imperfect information. In our approach, a deductive database is seen as a pair $\Delta = (P, F)$, where $P$ is essentially a datalog program with negation and $F$ is a set of facts. The facts recorded in the database are of several kinds:

- facts that express exact information, such as "an ostrich is a bird"
- facts that express uncertain information such as "an ostrich possibly doesn’t fly"
- facts that refer to extensional predicates
- facts that refer to intensional predicates.

In order to model such databases, we extend Kleene’s three-valued logic (true, false and unknown) by adding new values representing various degrees of truth. We structure these values by defining two orders, the truth order and the knowledge order. We then extend Przymusinski’s three-valued stable semantics [21] to a multivalued one, along the lines of Fitting [6]. We use these extension for describing the semantics of queries and updates in databases with uncertain information.

Motivation for this work comes from problems arising in query answering that combines information from multiple sources. A typical example is a multimedia information system, where a query can access data in a number of different subsystems (sound-, image-, text-subsystem etc.). Each subsystem provides the answer to a subquery and the system must then combine these answers in order to provide the answer to the overall query.

Let us illustrate our discussion using an example. Consider an application of a store that sells compact disks and assume that we want to produce a list of albums by Oscar Peterson, sorted according to their jazziness. To do this we use two sources of information:

- A relation $A(AlbumNo, Artist)$ residing on a relational database, and
- The estimates of an expert as to the type (jazzy, funky, etc) of the music announced on the cover of the albums. Let us think of these estimates as a relation $E(AlbumNo, Type)$ residing on a sound-subsystem.

The question arising here is how to combine the imperfect information provided by the sound-subsystem with the exact one provided by the relational database. A natural solution would be to assign the value 1 to all tuples in the relation $A$ and -1 to all tuples in the complementary relation (i.e. not stored in $A$), and to combine a tuple value with an estimate of the expert by using min operation. Various functions that may be used to define logical operations for combining uncertain information are discussed in [5].

In the light of our discussion, there are applications where we need (a) more than two logical values and (b) more that one order over these values (i.e. more than one way of structuring the set of logical values). Indeed, the values can be ordered w.r.t. their degree of truth (as for instance -1 and 1, being seen as false and true, are the least and the greatest values, respectively) or w.r.t. their degree of information or knowledge (as for instance 0, -1, and 1, being seen as nothing known or unknown, false and true, are the least, a maximal and another maximal values, respectively). Therefore we will introduce and...
use a multivalued logic with two orders, a truth order and a knowledge order.

Let us mention that there exist several formalisms on which are based various approaches tackling the matter of uncertain, incomplete and inconsistent information in logic programs and databases. These formalisms include the probability theory [20], [14], [13], [8], [23], [25], [15], the theory of fuzzy sets [24], [4], [2], the multivalued logics [11], [6], [18], [7], [16], [17], the possibilistic logic [2], the Dempster-Shafer theory of evidence [19], and hybrid (i.e. numerical and non numerical) formalisms [12], [1].

In the following Section II, we introduce formally the multivalued logic that we use, while in Section III we define programs and their multivalued semantics. In Section IV, we study extended deductive databases and their updating as well as data complexity of our multivalued semantics, and finally we provide some concluding remarks in Section V.

II. THE LOGIC $L_m$

The multivalued logic that we introduce and use in this paper comprises the usual values of the three-valued logic, i.e. true, false and unknown, but also additional values expressing various degrees of truth. As illustrated in the example provided in the introduction, we use numbers to represent these logical values, as follows: 1 for true, −1 for false, 0 for unknown, $\frac{m}{m}$, $\frac{(m-1)}{m}$ for different degrees of truthness or falseness, where $m$ is a positive integer.

The values 1 and −1 express exact information while values $0, \frac{1}{m}, \frac{2}{m}, \ldots, \frac{(m-1)}{m}$ express partial or uncertain information; particularly, the value 0 expresses the lack of information. We denote by $L_m$ the set of these $2m+1$ logical values, that is,

$$L_m = \{ \frac{n}{m} \mid n = 0, \ldots, m \}$$

As mentioned in the introduction, we consider two orders on the set $L_m$, the truth order that we denote by $\leq_t$, and the knowledge order that we denote by $\leq_k$. The truth order shows the degree of truth, and the knowledge order shows the degree of knowledge (or of information). The logical values of $L_m$ are thus ordered as follows:

- truth order: $-1 \leq t \leq_n m \leq n \leq 1$
- knowledge order: $0 \leq k \leq_n m \leq k \leq 1$ and $0 \leq k \leq n \leq k$

Note that $L_1$ is isomorphic with the well known Kleene’s three-valued logic. On the other hand $L_2$ is a five-valued logic having two new values w.r.t. $L_1$, namely 1/2 and −1/2, which, in the context of the logic $L_2$, will be interpreted as possibly true and possibly false, respectively.

For illustrative purposes the logic $L_2$ with its two orders is represented in Figure 1, where the axes t and k show increase in the truth and in the knowledge order, respectively. We observe that the truth order is a linear order, i.e. any two values of $L_2$ are comparable. The knowledge order, however, is not a linear order. Note that, in the knowledge order, the value −1 expresses more information than −1/2, and 1 expresses more information than 1/2. Note also that, in the knowledge order, −1 is not comparable neither with 1 nor with 1/2, and −1/2 is not comparable neither with 1 nor with 1/2. We shall use logic $L_2$ as a context for our examples for illustrative purposes. Unless mentioned otherwise, from now on we refer to the general logic $L_m$.

In the truth order, we shall use the logical connectives $\land, \lor$ and $\neg$ that we define as follows (see [5] for various functions that may be adapted and used to define logical connectives):

$$l_1 \land l_2 = \min(l_1, l_2)$$

$$l_1 \lor l_2 = \max(l_1, l_2)$$

and $\neg l = -l$, for any $l, l_1$ and $l_2$ in $L_m$. The connectives $\land$ and $\lor$ are in fact the meet and join operations in the truth order, and it is not difficult to see that $(L_m, \land, \lor)$ is a complete lattice. In the knowledge order, we shall use only the meet operation $\land$ defined as follows:

$$l_1 \land l_2 = \begin{cases} 0, & \text{if } l_1 l_2 \leq 0 \\ l_1, & \text{if } l_1 l_2 \leq l_2 \\ l_2, & \text{if } l_2 l_1 \leq l_1 \text{ and } l_1 l_2 > 0 \\ l_1 l_2, & \text{if } l_1 l_2 > 0 \end{cases}$$

Here, $\leq_k$ is the usual ordering of the reals and $|l|$ denotes the absolute value of $l$. It is not difficult to see that $(L_m, \land)$ is a complete semilattice.

III. LOGIC PROGRAMS AND THEIR MULTIVALUED SEMANTICS

In the logic $L_m$ one can consider rules in which the literals in the body are connected by any of the connectives $\land, \lor$ or $\land$. However, for the purposes of this paper, we only consider rules in which the literals in the body are connected by the truth conjunction $\land$.

Thus the programs that we consider in this paper are essentially datalog programs with negation, in which the logical values of $L_m$ can appear as 0-arity predicate symbols in the bodies of rules. The only difference from usual datalog programs lies in the semantics used. Indeed, as we have just explained, we consider additional logical values that express uncertain information.

In this setting, we define the multivalued semantics of programs and deductive databases.

A. Programs

The programs that we consider are built from atoms, which are predicate symbols with a list of arguments, for example $Q(a_1, \ldots, a_n)$, where $Q$ is the predicate symbol. An argument can be either a variable or a constant, and we assume that each predicate symbol is associated with a nonnegative integer called its arity. A literal is either an atom or a negated atom. A negated atom is a negative literal; one that is not negated is a positive literal. A rule is a statement of the form $A \leftarrow B_1, B_2, \ldots, B_n$, where $A$ is an atom and each $B_i$ ($i = 1, \ldots, n$)
is either a literal or a value from \( L_m \), seen in this case as a 0-arity predicate. The atom \( A \) is called the head of the rule and \( B_1, B_2, ..., B_n \) is called the body of the rule.

A program is a finite set of rules, and a positive program is one in which there is no negated literal. For example, 
\[
P : \quad a \leftarrow b; \quad b \leftarrow c; \quad c \leftarrow 0, d; \quad d \leftarrow -1/2
\]
is a positive program, as \( P \) contains no negative literal (-1/2 is not a negative literal since it does not contain the symbol \( - \)). Following usual practice, we call a predicate symbol \( \text{intensional if it appears in the head of a rule, and extensional otherwise.} \)

The Herbrand universe of a program \( P \) is the set of all constants that appear in \( P \) (and if no constant appears in \( P \) then the Herbrand universe is assumed to be a fixed singleton). By instantiation of a variable \( x \) we mean the replacement of \( x \) by a constant from the Herbrand universe. If we instantiate all variables of an atom or of a rule, then we obtain an instantiated atom or rule. The Herbrand base of a program \( P \) is the set of all possible instantiations of the atoms appearing in \( P \). The Herbrand base of a program \( P \) is denoted by \( HB_P \).

B. Valuations and Models

Given a program \( P \), we define a valuation \( v \) to be any function that assigns to every atom of the Herbrand base a logical value from \( L_m \). We shall make use of two special valuations that we denote by \( 0 \) and \(-1 \). The valuation \( 0 \) assigns the value 0 to every atom in the Herbrand base; it is the “nothing known” valuation. The valuation \(-1 \) assigns the value \(-1 \) to every atom in the Herbrand base; it is the “all false” valuation.

Given a valuation \( v \), we extend it to elements of \( L_m \) (seen as predicates of arity 0), to literals and to conjunctions of literals and 0-arity predicates from \( L_m \) as follows:
\[
v(l) = 1, \quad \text{where} \quad l \text{ is any element of } L_m,
\]
\[
v(\neg A) = -v(A), \quad \text{where} \quad A \text{ is any instantiated atom},
\]
\[
v(B_1 \land B_2 \land ... \land B_n) = 1 \quad \text{if } n = 0 \quad \text{and}
\]
\[
v(B_1 \land B_2 \land ... \land B_n) = v(B_1) \land v(B_2) \land ... \land v(B_n) \quad \text{if } n > 0 \quad \text{where} \quad B_i \text{ are instantiated literals or elements from } L_m.
\]

Now, we can extend the two orders \( \leq_s \) and \( \leq_k \) (that we have seen earlier) to the set \( V \) of all valuations in a natural way: for any valuations \( u \) and \( v \), define
\[
u \leq_s v \quad \text{if } u(A) \leq_s v(A) \quad \text{for all } A \text{ in } HB_P
\]
\[
u \leq_k v \quad \text{if } u(A) \leq_k v(A) \quad \text{for all } A \text{ in } HB_P.
\]

It is then not difficult to see that, in the truth order, \( V \) becomes a complete lattice while, in the knowledge order, \( V \) becomes a complete semilattice.

In the truth order, we say that a valuation \( v \) satisfies an instantiated rule \( A \leftarrow B_1, B_2, ..., B_n \) if \( v(A) \geq_1 v(B_1) \land B_2 \land ... \land B_n \). This definition is natural, as it expresses the fact that if \( A \) is deduced from \( B_1, B_2, ..., B_n \) then \( A \) must be assigned a logical value greater than or equal to the value assigned to \( B_1 \land B_2 \land ... \land B_n \). Now, if a valuation \( v \) satisfies all possible instantiations of rules of a program \( P \), then \( v \) is called a model of \( P \). Given a program \( P \), we shall denote by \( P^* \) the set of all possible instantiations of rules of \( P \). Note that \( P^* \) is also a program (possibly much larger, in general, than \( P \)).

Definition 1: - Model. A model of a program \( P \) is a valuation \( v \) that satisfies every rule of \( P^* \).

Given a program \( P \), we define the immediate consequence operator of \( P \) to be a mapping \( \Phi_P : V \rightarrow V \) defined as follows: for every valuation \( u \) in \( V \), \( \Phi_P(u) \) is a valuation s.t. for every atom \( A \) of the Herbrand base, \( \Phi_P(u)(A) = \)
\[
\begin{cases}
    -1, & \text{if there is no rule in } P^* \text{ with head } A \\
    \text{lb}_l(A) \cup A \leftarrow C \in P^*, & \text{otherwise}.
\end{cases}
\]

Here \( \text{lb}_l \) denotes the least upper bound in the truth order and \( C \) is a conjunction of atoms of the form \( B_1 \land ... \land B_n \). That is, \( \Phi_P \) takes as argument a valuation \( u \) and returns as a result a valuation \( \Phi_P(u) \) computed as follows:
\[
\begin{align*}
&\quad \text{for every atom } A \text{ of the Herbrand base do} \\
&\quad \text{begin} \\
&\quad \Phi_P(u)(A) := -1; \\
&\quad \text{for every rule } A \leftarrow B_1, ..., B_n \text{ in } P^* \text{ do} \\
&\quad \Phi_P(u)(A) := [\Phi_P(u)(A)] \lor [u(B_1 \land B_2 \land ... \land B_n)] \\
&\quad \text{end}.
\end{align*}
\]

Note that, if there is no rule in \( P^* \) with head \( A \), then \( \Phi_P(u)(A) \) is assigned the value \(-1 \). We do this because (as in the case of well-founded semantics) we privilege the truth order. This means that, if an atom \( A \) is not the head of any rule (and thus we have no information on its truth value) then \( A \) is assigned the least value of the truth order, namely \(-1 \). As a consequence, we assign the least element of the truth order to every atom \( A \) that is not head of a rule.

C. Semantics of Positive Programs

We can show easily that if \( P \) is a positive program then its immediate consequence operator \( \Phi_P \) is monotone in the truth order. Now, as the set \( V \) of all valuations is a complete lattice (in the truth order), \( \Phi_P \) has a least fixpoint, denoted \( \text{lf}_{P^*} \Phi_P \). We can show that this least fixpoint is, in fact, the least model of \( P \). So we call \( \text{lf}_{P^*} \Phi_P \) the multivalued semantics of \( P \), or simply the semantics of \( P \). It follows that the semantics of \( P \) can be computed as the limit of the following sequence of iterations of \( \Phi_P \):
\[
\begin{align*}
&u_0 = -1 \\
&u_{n+1} = \Phi_P(u_n) \text{ for } n \geq 0.
\end{align*}
\]

Here, \(-1 \) is the valuation that assigns the value \(-1 \) to every atom of the Herbrand base.

Note that our programs contain no function symbols, so the computation of the semantics \( \text{lf}_{P^*} \Phi_P \) terminates in a finite number of steps. Note also that, due to the way the semantics is computed, i.e. by iterating the immediate consequence operator to reach its fixpoint or, equivalently, by repeatedly applying the rules until nothing new is obtained, the semantics can be intuitively interpreted as the total knowledge that can be deduced from the program.

D. Semantics of Programs with Negation

If the program \( P \) contains negative literals then the immediate consequence operator \( \Phi_P \) is no more monotone. As a consequence we can no more define the semantics of \( P \) as the least model of \( P \), since such a model may not exist.

So we have to look for a new definition of semantics for \( P \) extending the semantics of positive programs. The idea,
explained intuitively through the following example, is, again, to try to deduce all the possible knowledge from a program with negation.

Example 1: Consider the logic \( L_2 \) and the following program \( P \), where \( a, b, c \) and \( d \) are predicates of arity 0:

\[
\begin{align*}
\text{a} & \leftarrow \neg \text{b}; \\
\text{b} & \leftarrow \neg \text{c}; \\
\text{c} & \leftarrow \neg \text{a}; \\
\text{d} & \leftarrow 1/2; \\
e & \leftarrow \neg d.
\end{align*}
\]

Note that, as all predicates are of arity 0, the program \( P^* \) coincides with \( P \), i.e. \( P^* = P \).

**Step 0:** We begin by assuming “nothing known” about the negative literals of the program i.e., we begin with the valuation 0 for the negative literals. Now, if we replace all negative literals of \( P^* \) by their values under 0 (i.e. by 0), then we obtain the following positive program denoted by \( P/0 \):

\[
\begin{align*}
\text{a} & \leftarrow 0; \\
\text{b} & \leftarrow 0; \\
\text{c} & \leftarrow 0; \\
\text{d} & \leftarrow 1/2; \\
e & \leftarrow a, 0.
\end{align*}
\]

As \( P/0 \) is a positive program, we can compute its semantics applying repeatedly the immediate consequence operator \( \Phi_{P/0} \) (starting with the valuation 0 as explained earlier). We find the following model (represented by a table, where the atoms of the Herbrand base appear in the first row and their logical values in the second row):

\[
\begin{array}{cccc}
a & b & c & d \\
0 & 0 & 0 & 1/2 0
\end{array}
\]

As a result, we have increased our knowledge, since we now know that \( d \) is associated with 1/2 (as opposed to 0 that we had assumed initially). This increased knowledge is represented by the valuation \( v_1 = \Phi_{P/0} \).

**Step 1:** We can now repeat the process, using \( v_1 \) instead of 0. That is, we can now replace all negative literals of \( P^* \) by their new values under \( v_1 \), to obtain again a positive program that we shall denote by \( P/v_1 \):

\[
\begin{align*}
\text{a} & \leftarrow 0; \\
\text{b} & \leftarrow 0; \\
\text{c} & \leftarrow 0; \\
\text{d} & \leftarrow 1/2; \\
e & \leftarrow a, -1/2.
\end{align*}
\]

As \( P/v_1 \) is a positive program, we can compute its semantics using \( \Phi_{P/v_1} \) (as above):

\[
\begin{array}{cccc}
a & b & c & d \\
0 & 0 & 0 & 1/2 1/2
\end{array}
\]

As a result, we have increased our knowledge even further, since we now know that \( e \) is associated with \(-1/2\) (as opposed to 0 previously). This increased knowledge is represented by the valuation \( v_2 = \Phi_{P/v_1} \).

**Step 2:** If we repeat the process once more then we obtain the program \( P/v_2 \) and its least fixpoint:

\[
\begin{align*}
\text{a} & \leftarrow 0; \\
\text{b} & \leftarrow 0; \\
\text{c} & \leftarrow 0; \\
\text{d} & \leftarrow 1/2; \\
e & \leftarrow a, -1/2.
\end{align*}
\]

\[
\begin{array}{cccc}
a & b & c & d \\
0 & 0 & 0 & 1/2 1/2
\end{array}
\]

We observe that \( \Phi_{P/v_0} = v_2 \), so we can no more increase our knowledge. That is, the valuation \( v_2 \) represents all the knowledge that we can have from \( P \). This knowledge is: \( a, b \) and \( c \) are unknown, \( d \) is possibly true and \( e \) is possibly false.

The important thing to note, in the above example, is that: each step \( i \) takes as input a valuation \( v_i \) of \( P \), and returns as a result a valuation \( v_{i+1} = \Phi_{P/v_i} \) of \( P \), via a transformation of \( P \) to \( P/v_i \) followed by a semantics computation for \( P/v_i \). The transformation \( v_i \to v_{i+1} \) is denoted by \( GLP \), i.e. \( v_{i+1} = GLP(v_i) \), and is called the extended Gelfond-Lifschitz transformation of \( P \).

Let us now define formally the concepts illustrated by the previous example. Given a program \( P \) and a valuation \( v \), we denote by \( P/v \) the program obtained from \( P^* \) by replacing each negative literal \( \neg A \) of \( P^* \) by its value under \( v \), i.e. by \( v(A) \). It is important to note that: (1) the program \( P/v \) is a positive program and (2) its semantics (i.e. the \( \Phi_{P/v} \)) gives the information deduced from \( P \) by assuming that the values for the negative premises are given by \( v \).

**Definition 2:** The mapping \( GLP : V \to V \) defined by \( GLP(v) = \Phi_{P/v} \), for all \( v \) in \( V \), is called the extended Gelfond-Lifschitz transformation of \( P \). We can show that the transformation \( GLP \) is monotone in the knowledge order:

**Theorem 3:** The operator \( GLP \) is monotone in the knowledge order.

Now, as the set \( V \) of all valuations is a complete semilattice (in the knowledge order), \( GLP \) has a least fixpoint, denoted \( GLP(P) \). In the previous example, this least fixpoint is the valuation \( v_2 \). We can show that the fixpoints of \( GLP \) are models of \( P \), and we call them multivalued models of \( P \). The \( \Phi_{P} \) is the multivalued model of \( P \) that has the least degree of information. In fact, \( \Phi_{P} \) represents all the information that one can deduce from \( P \), as we have seen in the previous example. So we choose \( \Phi_{P} \) to represent the semantics of \( P \) and we call it the multivalued semantics of \( P \). It follows that the multivalued semantics of \( P \) can be computed as the limit of the following sequence of iterations of \( GLP \):

\[
w_0 = 0 \\
w_{n+1} = GLP(w_n)
\]

We note that if \( P \) is a positive program then we have that: \( P/v = P^* \) for any valuation \( v \), and thus \( GLP(v) = \Phi_{P/v} \) for any valuation \( v \). It follows that the semantics of programs with negation extends the semantics of positive programs.

We recall that Gelfond and Lifschitz have introduced 2-valued stable-model semantics [9], which was then extended to 3-valued stable semantics by Przymusinski [21]. We note that, if we use the three-valued logic \( L_3 \) then the multivalued semantics described here coincides with Przymusinski’s three-valued stable semantics [21] that, in turn, coincides with the well-founded semantics [22].

**IV. EXTENDED DEDUCTIVE DATABASES WITH UNCERTAIN INFORMATION**

**A. Databases and Their Semantics**

We have seen so far the definition of a program and its multivalued semantics, in the logic \( L_m \). Now, the rules of a program represent our general perception or knowledge of a part of the “real” world. Our general perception represented by the rules, is then confronted to the observation of “real” world facts.

Informally, a fact is a statement such as “an ostrich possibly can’t fly”, describing the results of our observation. More formally, a fact is an instantiated atom along with the logical value that observation assigns to it. As a consequence, we
shall represent facts as pairs of the form $< A, l >$, where $A$ is an instantiated atom and $l$ is any value from $L_m$. For instance, in logic $L_2$, the fact above will be represented as $< \text{flies(ostrich)}, -1/2 >$. What we shall call a database is a set of rules (i.e. a program) $P$, along with a set of facts $F$.

Definition 4: An extended deductive database or simply a database is a pair $\Delta = (P, F)$, where $P$ is a program, called also the intensional database, and $F$ is a set of facts, called also the extensional database. $\square$

Now, when we observe a fact $< A, l >$ and we place it in the database, we certainly intend to assign the logical value $l$ to $A$, no matter what value is assigned to $A$ by the multivalued semantics of $P$. In other words, the semantics of the database will be that of $P$ modified so that $A$ is assigned the value $l$. More formally, in order to define the semantics of a database $\Delta = (P, F)$, we first transform $P^*$, using $F$, as follows:

- Step 1: Delete from $P^*$ every rule whose head appears in a fact of $F$.
- Step 2: To the program thus obtained add a rule $A \leftarrow l$ for every fact $< A, l >$ in $F$.

Let us denote by $P/F$ the program obtained by applying the above steps 1 and 2 to $P^*$. Note that Step 1 removes every rule of $P^*$ that can possibly influence the value assigned to $A$ in the semantics of $\Delta$; and Step 2 guarantees that $A$ will actually be assigned the value $l$, provided of course that there is no other fact $< A, l' >$ in $F$ with $l \neq l'$. Hence the following definitions:

Definition 5: -Database Consistency. A database $\Delta = (P, F)$ is called consistent if $P$ does not contain two facts of the form $< A, l >$ and $< A, l' >$ with $l \neq l'$. $\square$

Definition 6: -Database Semantics. The semantics of a consistent database $\Delta = (P, F)$ is defined to be the multivalued semantics of $P/F$. $\square$

The following proposition says that the semantics just defined does meet our intentions:

Proposition 7: Let $\Delta = (P, F)$ be a database and let $v$ be the semantics of $\Delta$. Then, for every fact $< A, l >$ in $F$, we have $v(A) = l$. $\square$

We note that the graph of a valuation $v$ is a set of pairs of the form $< A, v(A) >$, i.e. a set of facts. In particular, the graph of the semantics of a database is a set of facts. We shall refer to these facts as the database facts.

B. Data Complexity

Data complexity, as defined by Vardi in [26] for a conventional deductive database, is the time complexity of the evaluation of a ground atomic query (that is, a query without variables), expressed w.r.t. the size of the extensional database (i.e. the set of facts). Note that the intensional database (i.e. the set of rules) is assumed to be fix, and only the extensional database may vary during updating. Obviously, data complexity depends on the semantics defined for the deductive database. We can easily adapt this definition to our approach.

Let $\Delta = (P, F)$ be an extended deductive database. An atomic query is defined as $q(A, l)$ where $q$ is the name of the query, $A$ is an atom and $l$ a logical value. The meaning of the query is “which tuples contained in the relation associated to the atom $A$ are assigned a logical value greater or equal to $l$ w.r.t. the truth (knowledge) order, in the semantics of the database?”. In particular, note that $q(A, 1)$ corresponds to a query in the conventional deductive database approach.

Data complexity in our framework is defined to be the time complexity of the evaluation of a ground atomic query $q(A, l)$ (that is, an atomic query in which the atom $A$ has no variables) in the multivalued semantics of the database $\Delta$ w.r.t. $|F|$, where $|F|$ stands for the size of the extensional database $F$. That is, evaluating this query means checking if the logical value assigned to $A$ in the semantics of the database is at least $l$ with respect to the truth (knowledge) order.

We can prove that the Herbrand base size is polynomially bounded w.r.t. $|F|$; moreover we can show that one application of the operators $\Phi_{P/F}$ and $GL_{P/F}$ costs polynomial time w.r.t. $|F|$, and further, that $I_{P/F}$ can be evaluated in polynomial time w.r.t. $|F|$. Thus the following result:

Theorem 8: The semantics of a database $\Delta = (P, F)$ can be computed in polynomial time w.r.t. the size of the set of facts $F$.

As a ground atomic query $q(A, l)$ can easily be evaluated against the semantics of $\Delta$ (this can be done also in polynomial time w.r.t. $|F|$), we have:

Theorem 9: The data complexity of the multivalued semantics is polynomial.

C. Database Updating

Informally, by updating a database $\Delta = (P, F)$ we mean adding a fact $< A, l >$ in the set $F$. Of course, the intention is that the atom $A$ must be assigned the value $l$ in the semantics of the updated database.

Example 2: Consider the logic $L_2$ and a database $\Delta = (P, F)$ defined by:

$$P: \quad a \leftarrow \neg b; \quad b \leftarrow c; \quad c \leftarrow b \quad \text{and} \quad F = \emptyset.$$ 

Let $v$ be the semantics of $\Delta$. As $F = \emptyset$, the semantics of $\Delta$ is that of $P$, i.e. $v(a) = 1$ and $v(b) = v(c) = -1$. Suppose now that we want to update the database by adding the fact $< a, 1/2 >$. Let $\Delta' = (P, F')$ be the updated database.

Intuitively, it is natural to consider as “meaning” for $\Delta'$ the following valuation $v'$: $v'(a) = 1/2$ (because we added the fact $< a, 1/2 >$) and $v'(b) = v'(c) = -1$. Informally, the reason why $a$ and $b$ remain false is because trying to “prove” $b$ we need to “prove” $c$ and trying to “prove” $c$ we need to “prove” $b$. The situation here is similar to that with unfounded sets of well-founded semantics (and here the set $\{b, c\}$ reminds us of an unfounded set). $\square$

More formally, database updating is defined as follows:

Definition 10: Let $\Delta = (P, F)$ be a consistent database. The update of $\Delta$ by a fact $< A, l >$, denoted by $upd(\Delta, A, l)$, is a new database $\Delta' = (P, F')$, where $F'$ is defined by:

1. remove from $F$ every fact $< A, l' >$ such that $l' \neq l$
2. to the result thus obtained, add $< A, l >$. $\square$

We note that, when $l = 1$, the operation $upd(\Delta, A, l)$ corresponds to “insert $A$ in $\Delta$” and when $l = -1$, the operation $upd(\Delta, A, l)$ corresponds to “delete $A$ from $\Delta$”. However, $l$ can have any other value from $L_m$. $\square$

International Scholarly and Scientific Research & Innovation 1(9) 2007 2854

ISNI:0000000091950263
D. Properties of Updates

Database updating, as defined here, enjoys certain properties that correspond to intuition. In order to state these properties, let us call two databases $\Delta$ and $\Delta'$ (with the same Herbrand base) equivalent if they have the same semantics. We shall denote this by $\Delta \equiv \Delta'$. The first property of updating is idempotence, as expressed by the following proposition:

**Proposition 11:** For any database $\Delta$, and for any instantiated atom $A$ and logical value $l$, we have:

$$upd(upd(\Delta, A, l), A, l) \equiv upd(\Delta, A, l).$$

The following property says that, under certain conditions, the order in which updates are performed is not important:

**Proposition 12:** For any database $\Delta$, and for any instantiated atoms $A$ and $A'$ and logical values $l$ and $l'$ we have:

$$upd(upd(\Delta, A, l, A', l'), A, l) \equiv upd(upd(\Delta, A', l, A, l), A, l)$$

and if $A$ and $A'$ are distinct atoms then

$$upd(upd(\Delta, A, l, A', l'), A', l') \equiv upd(upd(\Delta, A', l, A, l), A', l').$$

The following proposition states a property of reversibility for updates. Roughly speaking, this property means that if we modify the value of a database fact $<A,l>$ from $l$ to $l'$, and from $l'$ back to $l$, then we recover the original database.

**Proposition 13:** Let $\Delta$ be a database and let $<A,l>$ be a database fact. Then

$$upd(upd(\Delta, A, l'), A, l) \equiv \Delta.$$

Another property of updates is monotonicity. Roughly speaking, this property means that if the value of a database fact increases then so does the database semantics. In the knowledge order, this property holds for any databases:

**Proposition 14:** Let $\Delta = (P,F)$ be a database with semantics $v$, let $<A,l>$ be any fact, and let $v'$ be the semantics of the database $\Delta' = upd(\Delta, A, l)$. Then the following statements hold:

1. if $l \geq_k v(A) \text{ then } v' \geq_k v$
2. if $l \leq_k v(A) \text{ then } v' \leq_k v$.

In the truth order, however, monotonicity holds only for positive databases:

**Proposition 15:** Let $\Delta = (P,F)$ be a database with semantics $v$, let $<A,l>$ be any fact, and let $v'$ be the semantics of the database $\Delta' = upd(\Delta, A, l)$. If $P$ is a positive program, then the following statements hold:

1. if $l \geq t v(A) \text{ then } v' \geq t v$
2. if $l \leq t v(A) \text{ then } v' \leq t v$.

V. CONCLUSION

We have seen an approach in which deductive databases are extended in two ways. First, the database can contain uncertain information and, second, the database updating is deterministic over intentional predicates. In order to express uncertainty we have introduced a multivalued logic called $\mathcal{L}_m$, with a double algebraic structure of lattice and semilattice w.r.t. two orders - the truth order and the knowledge order, respectively. We have defined a multivalued semantics for programs and databases in the context of the logic $\mathcal{L}_m$. This semantics extends successful conventional semantics as the three-valued stable semantics and the well-founded semantics, and has a polynomial data complexity.

REFERENCES