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

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.

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).

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 of a relation $E(AlbumNo, Type)$ residing on a sound-subsystem. Each tuple in relation E is associated with a number, say a real between -1 and 1, indicating the expert's opinion as to whether the album is of the type: 1 (or -1) indicates that the expert thinks the music of the album is certainly (or is not at all, respectively) of the type; 0 expresses that the expert has no particular opinion (so no information available); 0.9 (or -0.9) means that the expert is highly confident that the music is (or is not, respectively) of the type. The expert could for instance be an approximate pattern matching algorithm that looks if words from the language of jazz occur in samples from the CD's.

The multimedia query Q that we want to answer is then a conjunct of two atoms as follows:

$$Q(X) \leftarrow A(X, oscarPeterson) \ \& \ E(X, jazz)$$

The question arising here is how we 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*, $\pm 1/m, \ldots, \pm(m-1)/m$ for different degrees of truthness or falseness, where m is a positive integer.

The values 1 and -1 express *exact* information while values $0, \pm 1/m, \ldots, \pm(m-1)/m$ 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,

$$L_m = \{\pm(n/m) \mid n = 0, \ldots, m\}$$

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: $-1 \leq_t -n/m \leq_t 0 \leq_t n/m \leq_t 1$
- knowledge order: $0 \leq_k -n/m \leq_k -1$ and $0 \leq_k n/m \leq_k 1$, for $n = 0, \ldots, m$.

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

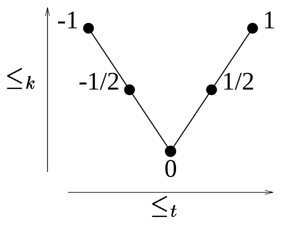

Fig. 1 The logic L_2

$-1/2$ 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 $\wedge, \vee$ and $\neg$ that we define as follows (see [5] for various functions that may be adapted and used to define logical connectives):

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

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

Here, $\leq$ is the usual ordering of the reals and $\mid l \mid$ denotes the absolute value of l. It is not difficult to see that $(L_m, \otimes)$ 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 $\wedge, \vee$ or $\otimes$. However, for the purposes of this paper, we only consider rules in which the literals in the body are connected by the truth conjunction $\wedge$.

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 $Q(a_1, .., a_n)$, 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 $A \leftarrow B_1, B_2, ..., B_n$, where A is an atom and each B_i $(i = 1, ..., n)$

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,

$P: \ a \leftarrow b; \quad b \leftarrow c; \quad c \leftarrow 0, d; \quad d \leftarrow -1/2$

is a positive program, as P contains no negative literal ($-1/2$ is not a negative literal since it does not contain the symbol $\neg$). Following usual practice, we call a predicate symbol *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* 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 $\mathbf{0}$ and $-\mathbf{1}$. The valuation $\mathbf{0}$ assigns the value 0 to every atom in the Herbrand base; it is the “nothing known” valuation. The valuation $-\mathbf{1}$ 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:

$v(l) = l$, where l is any element of L_m,

$v(\neg A) = \neg v(A)$, where A is any instantiated atom,

$v(B_1 \wedge B_2 \wedge ... \wedge B_n) = 1$ if $n = 0$ and

$v(B_1 \wedge B_2 \wedge ... \wedge B_n) = v(B_1) \wedge v(B_2) \wedge ... \wedge v(B_n)$, if $n > 0$, where the B_i's are instantiated literals or elements from L_m.

Now, we can extend the two orders $\leq_t$ 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

$u \leq_t v$ if $u(A) \leq_t v(A)$ for all A in HB_P and

$u \leq_k v$ if $u(A) \leq_k v(A)$ for all A 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 $v(A) \geq_t v(B_1 \wedge B_2 \wedge ... \wedge B_n)$. 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 $B_1 \wedge B_2 \wedge ... \wedge B_n$. 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, $\Phi_P(u)(A) =$

$$\begin{cases} -1, \text{ if there is no rule in } P^* \text{ with head } A \\ lub_t\{u(C) \mid A \leftarrow C \text{ in } P^*\}, \text{ otherwise.} \end{cases} .$$

Here lub_t denotes the least upper bound in the truth order and C is a conjunction of atoms of the form $B_1 \wedge ... \wedge B_n$. That is, Φ_P takes as argument a valuation u and returns as a result a valuation $\Phi_P(u)$ computed as follows:

```
for every atom $A$ of the Herbrand base do
  begin $\Phi_P(u)(A) := -1$;
    for every rule $A \leftarrow B_1, ..., B_n$ in $P^*$ do
      $\Phi_P(u)(A) := [\Phi_P(u)(A)] \vee [u(B_1 \wedge B_2 \wedge ... \wedge B_n)]$
  end.
```

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 Φ_P is monotone in the truth order. Now, as the set V of all valuations is a complete lattice (in the truth order), Φ_P has a least fixpoint, denoted $lfp_t \ \Phi_P$. We can show that this least fixpoint is, in fact, the least model of P. So we call $lfp_t \ \Phi_P$ 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 Φ_P:

$u_0 = -\mathbf{1}$

$u_{n+1} = \Phi_P(u_n)$ for $n \geq 0$.

Here, $-\mathbf{1}$ 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 $lfp_t\Phi_P$ 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 Φ_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:

$$a \leftarrow \neg b; \quad b \leftarrow \neg c; \quad c \leftarrow \neg a; \quad d \leftarrow 1/2; \quad e \leftarrow a, \neg d.$$

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 $\mathbf{0}$ for the negative literals. Now, if we replace all negative literals of P^* by their values under $\mathbf{0}$ (i.e. by 0), then we obtain the following positive program denoted by $P/\mathbf{0}$:

$$a \leftarrow 0; \quad b \leftarrow 0; \quad c \leftarrow 0; \quad d \leftarrow 1/2; \quad e \leftarrow a, 0.$$

As $P/\mathbf{0}$ is a positive program, we can compute its semantics applying repeatedly the immediate consequence operator $\Phi_{P/\mathbf{0}}$ (starting with the valuation $-\mathbf{1}$ 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):

$$lfp_t\ \Phi_{P/\mathbf{0}} = \begin{bmatrix} a & b & c & d & e \\ 0 & 0 & 0 & 1/2 & 0 \end{bmatrix}$$

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 $v_1 = lfp_t\ \Phi_{P/\mathbf{0}}$.

Step 1: We can now repeat the process, using v_1 instead of $\mathbf{0}$. 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:

$$a \leftarrow 0; \quad b \leftarrow 0; \quad c \leftarrow 0; \quad d \leftarrow 1/2; \quad e \leftarrow a, -1/2.$$

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

$$lfp_t\ \Phi_{P/v_1} = \begin{bmatrix} a & b & c & d & e \\ 0 & 0 & 0 & 1/2 & -1/2 \end{bmatrix}$$

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 $v_2 = lfp_t\ \Phi_{P/v_1}$.

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

$$a \leftarrow 0; \quad b \leftarrow 0; \quad c \leftarrow 0; \quad d \leftarrow 1/2; \quad e \leftarrow a, -1/2.$$

$$lfp_t\ \Phi_{P/v_2} = \begin{bmatrix} a & b & c & d & e \\ 0 & 0 & 0 & 1/2 & -1/2 \end{bmatrix}$$

We observe that $lfp_t\ \Phi_{P/v_2} = v_2$, 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 $v_{i+1} = lfp_t\ \Phi_{P/v_i}$ of P, *via* a transformation of P to P/v_i followed by a semantics computation for P/v_i. The transformation $v_i \rightarrow v_{i+1}$ is denoted by GL_P, i.e. $v_{i+1} = GL_P(v_i)$, 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 $v(\neg A)$. It is important to note that: (1) the program P/v is a positive program and (2) its semantics (i.e. the $lfp_t\ \Phi_{P/v}$) gives the information deduced from P by assuming that the values for the negative premises are given by v.

Definition 2: The mapping $GL_P : V \rightarrow V$ defined by $GL_P(v) = lfp_t \Phi_{P/v}$, for all v in V, is called the extended Gelfond-Lifschitz transformation of P. □

We can show that the transformation GL_P is monotone in the knowledge order:

Theorem 3: The operator GL_P is monotone in the knowledge order. □

Now, as the set V of all valuations is a complete semilattice (in the knowledge order), GL_P has a least fixpoint, denoted $lfp_k GL_P$. In the previous example, this least fixpoint is the valuation v_2.We can show that the fixpoints of GL_P are models of P, and we call them *multivalued models* of P. The $lfp_k GL_P$ is the multivalued model of P that has the least degree of information. In fact, $lfp_k GL_P$ represents all the information that one can deduce from P, as we have seen in the previous example. So we choose $lfp_k GL_P$ 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 GL_P:

$w_0 = \mathbf{0}$

$w_{n+1} = GL_P(w_n)$ for any $n \geq 0$.

We note that if P is a positive program then we have that: P/v=P^*, for any valuation v, and thus $GL_P(v)$=$lfp_t \Phi_{P^*}$, 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_1 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 $< flies(ostrich), -1/2 >$. 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. □

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 Δ; 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 F does not contain two facts of the form $< A, l >$ and $< A, l' >$ with $l \neq l'$. □

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

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 Δ. Then, for every fact $< A, l >$ in F, we have $v(A) = l$. □

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 Δ 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 polynomialy 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 $lfp_k GL_{P/F}$ 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 Δ (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: $a \leftarrow \neg b$; $b \leftarrow c$; $c \leftarrow b$ and $F = \emptyset$.

Let v be the semantics of Δ. As $F = \emptyset$, the semantics of Δ 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 Δ' 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). □

More formally, database updating is defined as follows:

Definition 10: Let $\Delta = (P, F)$ be a consistent database. The update of Δ 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 $l' \neq l$ and

(2) to the result thus obtained, add $< A, l >$. □

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

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 Δ and Δ' (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 Δ, 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 Δ, and for any instantiated atoms A and A' and logical values l and l' we have:

$upd(upd(\Delta, A, l), A', l) \equiv upd(upd(\Delta, A', l), A, l)$ and if A and A' are distinct atoms then

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

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 Δ 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 database:

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 $l \geq_k v(A)$ then $v' \geq_k v$

(2) if $l \leq_k v(A)$ 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 $l \geq_t v(A)$ then $v' \geq_t v$

(2) if $l \leq_t v(A)$ 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 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 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

[1] S. V. Alagar, F. Sadri, J. N. Said, "Semantics of an Extended Relational Model for Managing Uncertain Information", Proc. of International Conference on Information and Knowledge Management, 1995
[2] P. Bosc, H. Prade, "An Introduction to the Fuzzy Set and Possibility Theory-Based Treatment of Flexible Queries and Uncertain or Imprecise Databases", Uncertainty Management in Information Systems, Kluwer Academic Publishers, 1996
[3] D. Dubois, J. Lang, H. Prade, "Towards Possibilistic Logic Programming", Proc. of the Eight Intl. Conf. on Logic Programming, 1991
[4] M. H. van Emden, "Quantitative deduction and its fixpoint theory", J. Logic Programming, 3(1), 1986
[5] R. Fagin, "Combining fuzzy information from multiple systems", J. Computer and System Sciences, 58:1, 1999
[6] M. Fitting, "The Family of Stable Models", J. Logic Programming, 17(2,3&4), 1993
[7] M. Fitting, "Fixpoint semantics for logic programming - a survey", J. Theoretical Computer Science, 278(1-2), 2002
[8] N. Fuhr T. Rölleke "HySpirit – a Probabilistic Inference Engine for Hypermedia Retrieval in Large Databases", Proc. of 6th International Conference on Extending Database Technology, 1998
[9] M. Gelfond, V. Lifschitz, "The Stable Model Semantics for Logic Programming", Proc. of the Fifth Logic Programming Symposium, 1988
[10] A. Van Gelder, K.A. Ross, J.S. Schlipf, "The Well-Founded Semantics for General Logic Programs", J. ACM, 38(3), 1991
[11] M. Kifer, V. S. Subrahmanian, "Theory of Generalized Annotated Logic Programming and its Applications", J. Logic Programming, 12:(3-4), 1992
[12] V. S. Lakshmanan, F. Sadri, "Modeling Uncertainty in Deductive Databases", Proc. of 5th International Conference of Database and Expert Systems Applications, 1994
[13] V.S. Lakshmanan, N. Leone, R. Ross, V.S. Subrahmanian, "ProbView: a flexible probabilistic database system", J. ACM Transactions on Database Systems 22:3, 1997
[14] V.S. Lakshmanan, F. Sadri "Probabilistic Deductive Databases", Proc. of International Logic Programming Symp., 1994
[15] V. S. Lakshmanan, F. Sadri, "On a theory of probabilistic deductive databases", J. Theory and Practice of Logic Programming, 1(1), 2001
[16] Y. Loyer, N. Spyratos, D. Stamate, "Parameterised Semantics for Logic Programs - a Unifying Framework", J. Theor. Comput. Sci., 308(1-3), 2003
[17] Y. Loyer, N. Spyratos, D. Stamate, "Hypothesis-based semantics of logic programs in multivalued logics", J. ACM Transactions on Computational Logic 15(3), 2004
[18] B. Mobasher, D. Pigozzi, G. Slutzki, "Multi-valued logic programming semantics: An algebraic approach", J. Theor. Comput. Sci., 171:(1-2), 1997
[19] R. Ng, V. S. Subrahmanian, "Relating Dempster-Shafer Theory to Stable Semantics", Proc. of International Logic Programming Symp., 1991
[20] R. Ng, V. S. Subrahmanian "Probabilistic Logic Programming", J. Information and Computation, 101:2, 1992
[21] T.C. Przymusinski, "Extended Stable Semantics for Normal and Disjunctive Programs", Proc. of the Seventh Intl. Conference on Logic Programming, 1990
[22] T.C. Przymusinski, "Well-Founded Semantics Coincides with Three-Valued Stable Semantics", Fundam. Inform., 13, 1990
[23] F. Sadri, M. Bianco "On Equivalence of Queries in Uncertain Databases", Proc. of ACM International Conference on Information and Knowledge Management, 2000
[24] E.Y. Shapiro, "Logic Programs With Uncertainties: A Tool for Implementing Rule-Based Systems", Proc. of 8th International Joint Conference on Artificial Intelligence, 1983
[25] V.S. Subrahmanian, "Probabilistic Databases and Logic Programming", Proc. of 17th International Conference of Logic Programming, 2001
[26] M.Y. Vardi, "The complexity of relational query languages", Proc. of ACM Symp. on Theory of Computing, STOC, 1982