A logic for data and knowledge bases

C. Lecluse, N. Spyratos

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cédex
France
Tel.(1) 39 63 55 11

Rapports de Recherche

N° 606

A LOGIC FOR DATA AND KNOWLEDGE BASES

Christophe LECLUSE
Nicolas SPYRATOS

Février 1987

A LOGIC FOR DATA AND KNOWLEDGE BASES

UNE LOGIQUE POUR BASES DE DONNEES ET BASES DE CONNAISSANCES

Christophe LECLUSE
Nicolas SPYRATOS

Université de Paris-Sud
LRI (U.A. 410) Bat. 490
91405 ORSAY Cedex
FRANCE

ABSTRACT We present a class of first order languages suitable for data and knowledge representation. In support of this claim, we give algorithms for processing queries and updates in a restricted case corresponding to relational databases.

RESUME Nous présentons une famille de languages du premier ordre adaptée à la représentation des données et des connaissances. En particulier, nous présentons des algorithmes pour calculer la réponse aux requètes et effectuer les mises à jour dans un cas restreint correspondant aux bases de données relationelles.

November, 10 1986

1. INTRODUCTION

In the past few years, major attempts have been made to improve the power of database systems, in particular those based on the relational model of Codd [C70]. These attempts have concentrated, mainly, around the following issues:

(1) Non-procedural specification of queries (cf universal relation)

(2) The treatment of disjunctive information

(3) The addition of a deductive component (cf recursive queries)

(4) The inclusion of semantic constructs (cf ISA-hierarchies).

One way to accomodate all these issues in a common framework is to consider the relation schemes and the database constraints as sentences from predicate calculus, and the relations of the database as interpretations for these sentences. An interesting body of theory was developed around this approach, particularly with respect to special types of sentences about relations called dependencies (see [F82] and [FV84] for a survey of the area). In this paper, we adopt a different approach whereby the relation schemes, the database constraints, and the database, are all considered as strings of uninterpreted symbols. Interpretations for these symbols are provided using subsets of an underlying population of objects. Semantic implication is done essentially using set-containment, as explained (informally) in the following example.

Consider the (relational) database of Figure 1(a) which contains two tuples: the tuple ab (over scheme AB) and the tuple bc (over scheme BC). Define an interpretation to be any function φ assigning a subset of a given set ω to every symbol in the domains of A,B and C. An example is shown in Figure 1(b), where it is assumed that $\varphi(x) = \emptyset$ for all x not in {a,b,c} (moreover, it is assumed that ω is the set of integers). We extend an interpretation φ from atomic symbols to tuples as follows: for any tuple t, if $t=a_1 \ldots a_n$ then $\varphi(t)=\varphi(a_1) \cap \ldots \cap \varphi(a_n)$. Now, define an interpretation φ to be a model of our database, if φ assigns a nonempty set to every tuple appearing in the database. For example, the interpretation of Figure 1(b) is a model , as seen in Figure 1(c).

A B	B C	$\varphi(a) = \{1,2,3\}$	$\varphi(ab) = \varphi(a) \cap \varphi(b) = \{2,3\}$
a b	b c	$\varphi(b) = \{2,3,4\}$	$\varphi(bc) = \varphi(b) \cap \varphi(c) = \{4\}$
		$\varphi(c) = \{4,5\}$	
(a)		(b)	(c)

Figure 1

Consider now the following query on the database of Figure 1(a):

Q : List all tuples over scheme ABC

Define the answer $\alpha(Q)$ to be the set of all tuples t over ABC such that, t has a nonempty interpretation in every model of the database. Then the answer to query Q above is empty. Indeed, for the model φ of Figure 1(b), we find:

$$\varphi(abc) = \varphi(a) \cap \varphi(b) \cap \varphi(c) = \emptyset.$$

and, clearly, for any other tuple t over ABC such that $t \neq abc$, we can find a model φ' such that $\varphi'(t) = \emptyset$. It follows

that, if there is no restriction on models then $\alpha(Q)$ is empty. However, if we restrict the "admissible" models to those satisfying the constraint $\varphi(b) \subseteq \varphi(c)$, then we can infer that $\varphi(abc)$ is nonempty, in every admissible model. Indeed, in every model of the database, we have: $\varphi(ab) \neq \emptyset$, by definition of a model. On the other hand, $\varphi(b) \subseteq \varphi(c)$ implies that $\varphi(b) \cap \varphi(c) = \varphi(b)$. Combining these two facts, we obtain

$$\varphi(abc) = \varphi(a) \cap \varphi(b) \cap \varphi(c) = \varphi(a) \cap \varphi(b) = \varphi(ab) \neq \emptyset.$$

It follows that if $\varphi(b) \subseteq \varphi(c)$ then $\alpha(Q) = \{abc\}$. It is important to note that the closed world assumption [R83] is not needed in our model. Indeed, for any tuple containing symbols that do not appear in the database, we can always find a database model that falsifies this tuple. It follows that such tuples will never be included in an answer.

A restricted class of interpretations is obtained if we impose the following constraint: for every attribute A and for all a,a' in the domain of A, $\varphi(a) \cap \varphi(a') = \emptyset$. This class of interpretations, called partitioning interpretations, were first proposed and investigated in [S84], and subsequently used in [CKS85] to study the inference problem of so called partition dependencies.

The paper is organized as follows. In Section 2, we define a suitable first order language in which a database is defined as a set of formulas. In Section 3, we restrict our attention to databases without disjunctive data and we present algorithms for query answering, and update processing. Section 4 contains some concluding remarks. We assume some familiarity with first order logic and relational theory. The required definitions from logic are by now standard (see, for example, [L66]).

2. A DATABASE LOGIC.

For the purposes of formally defining a database, we restrict our attention to a suitable proper subset of first order languages, that we call database languages, and to a specific class of interpretations, that we call set interpretations. The resulting logic differs from that of Reiter [R83] in a significant way as explained earlier. As a consequence, we obtain finer (and, we believe, more natural) semantics, and algorithms for performing query answering and updating.

Database Languages

Define a first order language L to be a database language if L has the following properties:

0. There is a distinguished constant ø (that will be always interpreted by the empty set). The set of all constant symbols of L is infinite and denoted by *CONS*. The set of all variables of L is infinite and denoted by *VAR*.

1. There is only one binary function symbol ∘, that we call product. In the formation of terms we will omit "∘" when no confusion is possible. Thus if s and t are terms then we shall write 'st' instead of 's∘t'. The set of terms of L is denoted by *TERMS*. A term which is a constant or the product of constants is referred to as a tuple. The set of all tuples is denoted by *TUPLES*.

2. There are two binary relation symbols, $\approx$ which will function for us as set equality, and $\sqsubseteq$ which will function for us as set inclusion.

3. All other relation symbols of L are unary relation symbols and their set is denoted by *REL*. Among them, there is a distinguished non-empty finite subset U called the universe. The elements of U are called attributes. Let P_U be the set of all subsets of U. We assume a one-to-one function scheme from *REL* into $P_U \setminus \{\emptyset\}$ such that scheme(A)={A} for every attribute A in U. Intuitively, every symbol R of *REL* is a name for a set of attributes. Note, however, that different relation symbols are associated with different sets of attributes. We do not distinguish between a relation symbol and its scheme so for example, if scheme(R)={A,B}, we write {A,B}, or even AB instead of R. We also assume that there is a function type from *CONS* into P_U which associates a singleton to every constant of the language but ø, and type(ø)=$\emptyset$. We extend the function type to a function from TUPLES into P_U as follows: type(c.c')=type(c) $\cup$ type(c') for all tuples c and c'. For every relation symbol R in *REL*, we denote by dom(R) the set of all tuples t such that type(t)=scheme(R). Thus dom(R)= {t $\in$ TUPLES / type(t)=scheme(R)}. We assume that all domains are infinite sets. We shall often say "t is a tuple over R" instead of "t is in dom(R)".

Roughly speaking, the product constructs tuples from symbols, while $\approx$ and $\sqsubseteq$ are needed for writing special formulas expressing semantic information. We shall use the

following notation for typed quantifiers which is by now standard: for every symbol A in *REL*,

$\underline{\forall x/A\ f}$ stands for $\forall x\ Ax \supset f$ and

$\underline{\exists x/A\ f}$ stands for $\exists x\ Ax \wedge f$.

Set Interpretations

Roughly speaking, an interpretation φ of a database language L will assign to the terms t of L certain objects $\varphi(t)$ in a domain A, which the terms may be thought of as naming under the given interpretation. It will assign to each function symbol f of rank n a function $\varphi(f)$ of rank n, from the domain A into itself, and to each relation symbol r of rank n a relation $\varphi(r)$ of rank n on the domain A.

We want to think of an interpretation φ as attaching to each formula p some assertion about the domain A, which either holds or fails in A. The easiest way out is to take $\varphi(p)$ to be simply the value, truth or falsehood, of this assertion. Since we need not explain what is meant by thruth or falsehood, we choose instead two neutral objects, the numbers 1 and 0, to serve, respectively, instead of thruth or falsehood. We call 0 and 1 the truth-values, and denote by $B=\{0,1\}$ the set of those two thruth values. Then, if p is any formula, we shall take $\varphi(p)$ in B. It will be convenient now to require $\varphi(r)$, for r a relation symbol of rank n, to be not a set of n-tuples, but rather a function of rank n from A into B.

Let ω be a fixed set of (unspecified) objects and $P_\omega=\{\sigma \mid \sigma\subseteq\omega\}$ be the set of all subsets of ω. We shall take $A=P_\omega\times P_U$ as the interpretation domain in every database language L. It is important to note that ω can be any set.

However, in our discussions, we shall conveniently think of ω as being the set of nonnegative integers.

Precisely, an interpretation φ of L is a function defined on function symbols, relation symbols, terms and formulas with values as follows:

Function symbols

The only function symbol $\circ$ is interpreted as follows:

$\varphi(\circ)\ ((s,t)\ ,\ (s',t')) = (s \cap s', t \cup t')$

Relation symbols

The relation symbols of L are interpreted as follows: for all R in *REL*,

$\varphi(R)\ ((s,t))=1$ iff $type(t)=scheme(R)$.

$\varphi(\simeq)\ ((s,t),(s',t'))=1$ iff $s=s'$.

$\varphi(\sqsubseteq)\ ((s,t),(s',t'))=1$ iff $s \sqsubseteq s'$.

Terms

The values of φ on the set *CONS* $\cup$ *VAR* are in the domain $P_\omega \times P_U$. The projections of φ, denoted by φ_ω and φ_U are required to satisfy the following conditions:

- $\varphi_\omega(\varnothing) = \emptyset$.
- $\varphi_U(c)=type(c)$ for every c in *CONS*
- For every attribute A, the restriction of φ to dom(A) is injective. (this property corresponds to the unique names assumption).

We extend the function φ from *CONS* $\cup$ *VAR* to *TERMS* as follows: $\varphi(c \circ c') = (\varphi_\omega(c) \cap \varphi_\omega(c'), \varphi_U(c) \cup \varphi_U(c'))$ for all terms c and c' (which is in accordance with the interpretation of the product $\circ$ seen earlier).

Formulas

If R is a symbol of *REL* and s,t are terms then

$\varphi(Rt)=1$ iff $type(t)=scheme(R)$

$\varphi(t\approx s)=1$ iff $\varphi_\omega(t)=\varphi_\omega(s)$

$\varphi(t\sqsubseteq s)=1$ iff $\varphi_\omega(t)\subseteq\varphi_\omega(s)$

$\varphi(true)=1$, $\varphi(false)=0$

$\varphi(\neg p)=1-\varphi(p)$

$\varphi(p\wedge q)=\min(\varphi(p),\varphi(q))$, $\varphi(p\vee q)=\max(\varphi(p),\varphi(q))$

$\varphi(p\supset q)=\varphi(\neg p\vee q)$

Let x be a variable and p a formula, and define $I(x,\varphi)$ to be the set of all interpretations φ' that agree with φ except possibly on x: then

$\varphi(\forall xp)=1$ iff $\varphi'(p)=1$ for all φ' in $I(x,\varphi)$ and

$\varphi(\exists xp)=1$ iff $\varphi'(p)=1$ for some φ' in $I(x,\varphi)$.

Databases Defined

Let L be a database language. A database over L is a finite set of closed formulas of L.

Given a database D over L, we refer to the set of formulas of D without variables as data, and to the set of formulas of D with variables as knowledge. Accordingly, we denote data by d and knowledge by k. Thus, $D=d\cup k$. For example, consider the following database over L:

$$D=\{\neg ab\approx\emptyset,\ \neg a'b\approx\emptyset\vee\neg ab'\approx\emptyset,\ \neg bc\approx\emptyset\vee\neg b'c\approx\emptyset,\ \forall x/A\ \exists y/B\ x\sqsubseteq y\}$$

Assuming $U=\{A,B,C\}$, $scheme(R)=\{A,B\}$, $scheme(S)=\{B,C\}$, and a,a',b,b',c to be constants such that $type(a)=type(a')=\{A\}$, $type(b)=type(b')=\{B\}$, and $type(c)=type(c')=\{C\}$, we have:

$d = \{\neg ab\approx\emptyset,\ \neg a'b\approx\emptyset\vee\neg ab'\approx\emptyset,\ \neg bc\approx\emptyset\vee\neg b'c\approx\emptyset\}$

and $k = \{\forall x/A\ \exists y/B\ x\sqsubseteq y\}$

Let us note that, as databases are sets of formulas, all notions related to semantic implication are also applicable to databases. Thus, given a database D over L, an interpretation φ of L is a <u>model</u> for D if φ verifies all formulas of D (i.e. if $\varphi(p)=1$ for all p in D). If there is no model for D then we say that D is <u>inconsistent</u>, and otherwise that D is <u>consistent</u>. Moreover, we define the relation of <u>semantic implication</u>, $D \models D'$, between two databases, to hold iff every model for D is also a model for D'. We say that two databases D and D' are <u>semantically equivalent</u> (denoted by $D|\!=\!|D'$) if $D|\!=\!D'$ and $D'|\!=\!D$. Clearly, in these definitions we tacitly assume a specified class I of interpretations, and we should in principle write $D|\!=_I\!D'$. In practice, we shall commonly omit the subscript.

<u>Queries</u>

Queries are defined relative to a given database language L. Specifically, a <u>query for L</u> is any expression of the form <Rx|f> where:

- R is a relation symbol of *REL*, and x is a variable.
- f is a formula of L whose free variable is x.

The answer to a query for L is defined relative to a given database over L. Specifically, given a query Q=<Rx|f> and a database D, an <u>answer</u> to Q <u>with respect</u> to D is any tuple t such that D implies Rt, $\neg t=\emptyset$, and f[x/t]. The reason why we ask that $\neg t=\emptyset$ is the following: intuitively, a tuple $t=a_1 \ldots a_k$ is meaningfull if there exists an object in ω having property a_1 and ... and property a_k, that is, if $\varphi_\omega(t)=\varphi_\omega(a_1) \cap \ldots \cap \varphi_\omega(a_k)\neq\emptyset$ (here we view $\varphi_\omega(a_i)$ as the set of objects of ω having property a_i). The set of all answers to Q with respect to D is denoted by $\alpha(Q,D)$. Thus,

$$\alpha(Q,D) = \{\ t\ /\ D|{=}Rt \wedge \neg t{\approx}\emptyset \wedge f[x/t]\}.$$

Two databases D,D' over L are called query equivalent, denoted by $D{\equiv}D'$ iff $\alpha(Q,D){=}\alpha(Q,D')$, for every query Q. We say that D contains less information than D', denoted by $D \prec D'$, if $\alpha(Q,D){\subseteq}\alpha(Q,D')$ for every Q. Let us note that $D|{=}|D'$ implies $D{\equiv}D'$ but the inverse is not true. For example, if we consider the databases $D{=}\{\neg abc{\approx}\emptyset\}$ and $D'{=}\{\neg abc{\approx}\emptyset, \forall x/A\ \exists y/B\ \ x \sqsubseteq y\}$, we clearly have $D{\equiv}D'$ but not $D|{=}|D'$. We shall denote by BASES the set of all databases over the same language L. Clearly, the relation $\equiv$ is an equivalence relation on the set BASES. We denote by BASES/$\equiv$ the set of all equivalence classes of the set BASES. We can also define an order relation between equivalence classes as follows: $C{\leq}C'$ iff there is D in C and D' in C' such that $D \prec D'$. It is important to note that the databases of an equivalence class are just different representations of the same information. We assume that the user cannot (and should not) distinguish between equivalent databases.

Updates

Following Fagin et al [FUV83], we view the database "not merely as a collection of atomic facts, but rather as a collection of facts from which other facts can be derived". We shall therefore consider the updating of equivalence classes rather than of particular databases. For example, given the database $D{=}\{\neg abc{\approx}\emptyset\}$, we allow for the insertion of $\neg ab'{\approx}\emptyset$. Now, inserting over AB in D can be considered as a "view updating". We are interested in the information contained in the database and not in any

particular representation of this information. It follows that, by updating equivalence classes, we also allow for view updating.

3. SIMPLE DATABASES

Databases can be classified with respect to the type of data and to the type of knowledge that they contain. In its simplest form, the data contains no disjunction, that is, the data is a set of formulas of the form $\neg t \approx \emptyset$. Knowledge, on the other hand, may consist of two types of formulas: those that contain a single relation symbol, and those that contain more than one relation symbol. We refer to the former as interpretation constraints and to the latter as dependencies. We give here some examples of both with appropriate abbreviations for later use.

Interpretation Constraints: for all A in U:

Partition Constraint : $\forall x/A\ \forall x'/A \neg xx' \approx \emptyset \supset x \approx x'$.

Inclusion Constraint : $\forall x/A\ \forall x'/A \neg xx' \approx \emptyset \supset x \sqsubseteq x' \vee x' \sqsubseteq x$.

Transitive Constraint :

$\forall x/A\ \forall y/A\ \forall z/A\ \neg xy \approx \emptyset \wedge \neg yz \approx \emptyset \supset \neg xz \approx \emptyset$.

Filter Constraint : $\forall x/A\ \forall y/A \neg xy \approx \emptyset \supset \exists z/A\ z \sqsubseteq xy$.

Dependencies

Letting X and Y be relation symbols, the following are examples of dependencies:

XfunY stands for $\forall x/X\ \forall y/Y\ \forall y'/Y\ \neg xy \approx \emptyset \wedge \neg xy' \approx \emptyset \supset y \approx y'$.

$X \rightarrow Y$ stands for $\forall x/X\ \forall y/Y \neg xy \approx \emptyset \supset x \sqsubseteq y$.

X isa Y stands for $\forall x/X\ \exists y/Y\ x \sqsubseteq y$.

It is interesting to note that, under the partition constraint, the above three dependencies correspond all to the same function. Indeed, in all three cases, the set of pairs $\{(x,y)/\neg xy\approx\emptyset\}$ is a function. However, the three dependencies convey different semantic information. Moreover, it is not difficult to see that X isa Y $\supset$ X$\rightarrow$Y and X$\rightarrow$Y $\supset$ XfunY. The following proposition shows that, in a restricted case, the consistency of a database is the same in all three cases.

Proposition 1 Let R be a relation symbol of L, and d a set of formulas of the form $\neg t\approx\emptyset$ where t is a tuple of dom(R). Let X and Y be two relation symbols of L, such that scheme(X)$\subseteq$scheme(R) and scheme(Y)$\subseteq$scheme(R). The following properties are equivalent:

(1) d $\cup$ {X fun Y, Partition Constraint} is consistent.

(2) d $\cup$ {X $\rightarrow$ Y, Partition Constraint} is consistent.

(3) d $\cup$ {X isa Y, Partition Constraint} is consistent.

Proof

We can assume, without loss of generality, that scheme(R)={A,B}, scheme(X)={A} and scheme(¥)={B}. The two implications above show that (3) implies (2) and (2) implies (1) so it is enough to show that (1) implies (3). If d $\cup$ {X fun Y} is consistent, then we build a model m of d $\cup$ {X isa Y} as follows: we associate to every formula $\neg t\approx\emptyset$ of d a unique integer i_t. For every symbol x in dom(A)$\cup$dom(B), m(x) is the set of all integers i_t such that x appears in t. For every formula $\neg t\approx\emptyset$ of d such that t=ab, we change m(b) in m(a)$\cup$m(b). m clearly satifies the partition constraint for the attribute A. If there are b and b' in dom(B) such that m(b)$\cap$m(b')$\neq\emptyset$, then we added the same m(a) to m(b) and m(b') because the sets m(a) are pairwize disjoint. So, we have two formulas $\neg t\approx\emptyset$ and $\neg t\approx\emptyset$ in d, so Property (1) is false. Otherwise, m satisfies the partition constraint and also satisfies the formulas of d and the dependency X isa Y. So m is a model of d $\cup$ {X isa Y}. We conclude that (1) implies (3).

Define a database D to be simple if the data is a set of formulas of the form $\neg t \approx \emptyset$, and the knowledge consists of the partition constraint and dependencies of the form X isa Y. There is a correspondence between a simple database and a relational database if we view each tuple t as a relational tuple and each formula X isa Y as the relational functional dependency $X \rightarrow Y$. Moreover, it is shown in [CKS85] (in a somewhat different formalism) that the database D is consistent (i.e. has at least one model) iff the corresponding relational database is consistent.

Let D be a simple database. We denote by Σ the set of dependencies of D and by M(D) be the set of all models of D. From now on, we consider implication with respect to the class M(D). If m is a model of D and t is a term such that $\neg t \approx \emptyset$ is true in m, then we say that "t is true in m". Similarly, if $D \models \neg t \approx \emptyset$ then we say that "D implies t", or that "t is true with respect to D". Moreover, when we represent a database, we write t instead of $\neg t \approx \emptyset$. For example, Figure 2, below, represents a database with the following data $d=\{\neg a_1 b_1 \approx \emptyset, \neg a_2 \approx \emptyset\}$. Given a model m of D, we denote by T(m) the set of all tuples t that are true in m. We can define a partial order on M(D) as follows: $m \leq m'$ iff $T(m) \subseteq T(m')$. Let us define $T(D) = \cap \{T(m) \ / \ m \in M(D)\}$. Clearly, T(D) is the set of all tuples which are true in every model of D, hence for every tuple t, $D \models \neg t \approx \emptyset$ iff $t \in T(D)$. Finally, given a database D, we denote by *CONS*(D) the set of constants of *CONS* that appear in the database, and by TUPLES(D) the set of all tuples that can be built using the constants of *CONS*(D).

3.1 Minimal models - Query processing.

Let us recall that a query is defined with respect to a database language L and that it is an expression Q=<Rx/f> such that:

- R is a relation symbol of *REL*, and x is a variable.
- f is a formula of L whose free variable is x.

The answer to Q with respect to a database D denoted by $\alpha(Q,D)$ was defined to be the set of all tuples t such that D implies Rt, $\neg t \approx \varnothing$, and f[x/t]. Thus, in order to process queries in simple databases, we first have to solve the following inference problem. Given a simple database D, determine the set T(D), that is, determine the set of tuples implied by D. We say that a model m of D is a <u>query model</u> of D if

- m is a minimal model of D and
- $T(D)=T(m)\cap TUPLES(D)$.

The following example shows that minimality alone is not enough to characterize T(D). Indeed, in Figure 2, we can see a database D and two minimal models of D that are incomparable. The model m_1 is a query model of D whereas the model m_2 is not. Indeed, the tuple a_2b_1 is true in m_2 although a_2b_1 is not true with respect to D.

U={A,B}, dom(A)={$a_1,a_2,\ldots$}, dom(B)={$b_1,b_2\ldots$},

d:

<u>A B</u>	<u>A</u>
a_1b_1	a_2

k={A isa B}

m_1: $a_1\rightarrow\{1\}$, $a_2\rightarrow\{2\}$

m_2: $a_1\rightarrow\{1\}$, $a_2\rightarrow\{2\}$

$b_1 \rightarrow \{1\}$ $\qquad$ $b_1 \rightarrow \{1,2\}$

$b_2 \rightarrow \{2\}$

Figure 2

Below, we give an algorithm that computes a query model for a given simple database. In order to simplify the presentation of the algorithm, we assume that:

- All dependencies of Σ are of the form X isa A where X is a relation symbol and A a single attribute.
- Queries are of the form <Rx/true>.

Moreover, we consider only the 'semantic' component of an interpretation, that is we only consider its projection on P_ω.

Algorithm 1 Query Model

Input : A simple database D.

Output: A query model m_q of D (if D is consistent).

(1) Assign a distinct positive integer i_t to every tuple t of the database. Set $m_1(a)=\emptyset$ for every constant a.

(2) For every tuple $t=a_1 \ldots a_k$ of the database, add the integer i_t to $m_1(a_i)$, for i in $\{1,\ldots,k\}$

(3) Compute m_{i+1} from m_i as follows until $m_{i+1}=m_i$.

if there is X isa A in Σ, and x in dom(X) such that $m_i(x)\neq\emptyset$ and $m_i(a)\cap m_i(x)=\emptyset$, for all a in dom(A),

then m_{i+1} is obtained from m_i by taking a constant a^* in dom(A) such that $m_i(a^*)=\emptyset$ and setting $m_{i+1}(a^*)=m_i(x)$.

else if there is X isa A in Σ, x in dom(X) and a in dom(A) such that $m_i(a)\cap m_i(x)\neq\emptyset$,

then let $m(a)=m_i(a)\cup m_i(x)$, and $m(b)=m_i(b)$ if $b\neq a$. If there is a' in dom(A) such that $m(a)\cap m(a')\neq\emptyset$ and a or a' is not in *CONS*(D) (a' for instance), then $m_{i+1}(a)=m(a)\cup m(a')$, $m_{i+1}(a')=\emptyset$, and $m_{i+1}(b)=m_i(b)$ for $b\neq a,a'$. Otherwise, $m_{i+1}=m$.

else $m_{i+1}=m_i$.

(4) m_q is the last function m_{i0} computed at step 3.

Theorem 1 Algorithm 1 always terminates and if m_q satisfies the partition constraint, then m_q is a query model of D else D is inconsistent.

Proof

We can notice that, while computing m_{i+1} from m_i, following algorithm 1, the number of 'true' tuples is strictly increasing if $m_i\neq m_{i+1}$. If the algorithm does not terminate then we have an infinite increasing sequence of sets of tuples. As the number of constants in the database is finite, the number of tuples that may be built with them is also finite. So, in such a sequence, we have infinitely many new constants in the domain of at least one attribute. Moreover, the values of m_j for these new constants always stay pairwise disjoint. But, if a^* is a new constant of type A introduced to compute m_{i+1} from m_i, we have $m_{i+1}(a^*)=\{k_{a^*}\}\cup m_i(x)$ for some tuple x, so k_x is in $m_{i+1}(a^*)$. If x also contains a new constant b^*, we have $m_j(b^*)=\{k_{b^*}\}\cup m_j(y)$, for some term y and $j\leq i$. In this way, we can show that $m_{i+1}(a^*)$ contains at least one k_t where t is a tuple without any new constant. As there are only finitely many such tuples, the new constants cannot be pairwise disjoint. So the Algorithm terminates. Now, we have to show the following properties:

(1) If m_q verifies the partition constraint, then $T(D)=T(m_q)\cap TUPLES(D)$ and

(2) If m_q does not verify the partition constraint, then D is inconsistent.

- In order to show (1), we show that the property $T(m_i) \cap TUPLES(D) \subseteq T(D)$ is true for all i. If it is the case then, as m_q is a m_{j0} and m_q is a model of D, we clearly have $T(m_q) \cap TUPLES(D) = T(D)$. We clearly have $T(m_1) \cap TUPLES(D) \subseteq T(D)$. If $T(m_i) \cap TUPLES(D) \subseteq T(D)$ and X isa A,a and x are the elements used to build m_{i+1}. m_{i+1} differs from m_i only for the value $m_i'(a)$. If t is a tuple such that $m_{i+1}(t) \neq \emptyset$ and $m_i(t) = \emptyset$, then a is a sub-tuple of t and we can write t=at'. Then we have $m_{i+1}(t) = m_{i+1}(a) \cap m_{i+1}(t') = (m_i(a) \cup m_i(x)) \cap m_i(t') = m_i(x) \cap m_i(t')$ and so we have $m(x) \cap m(t') \neq \emptyset$ and $m(a) \cap m(x) \neq \emptyset$ for every model m of D. As m satisfies the dependency X isa A, we have $m(x) \subseteq m(a)$ so $m(t) \neq \emptyset$ for every model m of D. So $T(m_{i+1}) \cap TUPLES(D) \subseteq T(D)$. If a new constant a^* is introduced then $T(m_{i+1}) \cap TUPLES(D) = T(m_i) \cap TUPLES(D)$.

- If m_q does not satisfy PART, let i be the first integer such that $m_1 \ldots m_{i-1}$ satisfy PART but not m_i. Let X isa A, x et a be the elements used to compute m_i from m_{i-1}. Using (1), we can show that there are two tuples xa et xa' in T(D) which is impossible if D is consistent. So D is inconsistent.

Example 1 Let D be the following simple database: there are three attributes A,B and C, and three relation symbols R,S and T with schemes {A,B},{B,C} and {A,B,C} repectively. The data are $\neg ab \approx \emptyset$, $\neg bc \approx \emptyset$ and $\neg b'c' \approx \emptyset$. The only dependency is B isa C . Figure 3 shows a query model of D as computed by Algorithm 1:

m_q: a $\rightarrow$ {1}

b $\rightarrow$ {1,2} b' $\rightarrow$ {2}

c $\rightarrow$ {1,2} c' $\rightarrow$ {2}

Figure 3

We turn now our attention to query processing. Given a database, our general plan works as follows: Produce a

query model m_q of the database and for each query expressed in terms of the database language,

(a) Process the query using this model

(b) Give the result in terms of the database language.

In this way, the syntactic component, namely the database language, serves as an interface in which the user interacts with the system, while the semantic component, namely the query model, serves as an environment in which the actual processing takes place. Indeed, given a query Q, the answer $\alpha(Q,D)$ is the set of all tuples t over Q such that t is in TUPLES(D) and $m_q(t)\neq\emptyset$. For instance, in Example 1 above, for Q=<Tx/true>, we find $\alpha(Q,D)=\{abc\}$, because abc is the only tuple over T such that abc is in TUPLES(D) and $m_q(abc)\neq\emptyset$.

3.2 Characteristic models - update processing

In order to give algorithms for performing updates in a simple database D, we have to solve the following inference problems:

(1) Given a simple database D, determine the set T(D), that is, determine the set of tuples implied by D (which is the inference problem solved for queries), and

(2) Determine whether $D\models s\sqsubseteq t$, for tuples s and t in T(D).

We solve these problems by constructing a query model m_0 such that:

(a) If $m_0(t_0)\subseteq m_0(t_1)\cup\ldots\cup m_0(t_n)$ for some tuples $t_0,\ldots,t_n$ then we have $D\models t_0\sqsubseteq t_i$, for some i in $\{1,\ldots,n\}$.

(b) There is no pair of inseparable integers in m_0. (We say that two integers i and j are inseparable in m if there is a constant a such that i and j are in m(a) and, for every constant symbol b, i is in m(b) iff j is in m(b)).

Such a model allows us to answer the questions above and we call it a characteristic model. The following proposition gives an alternative formulation of property (a) above.

Proposition 2 For any query model m_0, the following properties are equivalent:

(1) For all $t_0,\ldots,t_n$ in $T(D)$, if $m_0(t_0) \subseteq m_0(t_1) \cup \ldots \cup m_0(t_n)$ then there is i in $\{1,\ldots,n\}$ such that $D \models t_0 \sqsubseteq t_i$.

(2) $\forall t \in T(D)\ \exists k_t \in m_0(t)\ \forall s \in T(D)\ k_t \in m_0(s) \Rightarrow D \models t \sqsubseteq s$.

Proof

Easy.

The following algorithm computes a characteristic model of a simple database D. As in the previous section, we assume that all dependencies of Σ are of the form X isa A where X is a relation symbol and A a single attribute. Moreover, we consider only the "semantic" component of an interpretation, that is, we only consider its projection on P_ω.

Algorithm 2 Characteristic Model

Input : A simple database D.

Output: A characteristic model m_0 of D (if D is consistent).

(1) Assign a distinct positive integer i_t to every sub-tuple t of a tuple of the database. Set $m_1(a)=\emptyset$ for every constant a.

(2) For every sub-tuple $t=a_1...a_k$ of a tuple of the database, add the integer i_t to $m_1(a_i)$, for i in $\{1,...,k\}$

(3) Compute m_{i+1} from m_i as follows until $m_{i+1}=m_i$.

-if there is X isa A in Σ, and x in dom(X) such that $m_i(x)\neq\emptyset$ and $m_i(a)\cap m_i(x)=\emptyset$, for all a in dom(A).

then m'_i is obtained from m_i by taking a constant a^* in dom(A) such that $m_i(a^*)=\emptyset$ and setting $m'_i(a^*)=m_i(x)$.

else if there is X isa A in Σ, x in dom(X) and a in dom(A) such that $m_i(a)\cap m_i(x)\neq\emptyset$,

then let $m(a)=m_i(a)\cup m_i(x)$ and $m(b)=m_i(b)$ if $b\neq a$. If there is a' in dom(A) such that $m(a)\cap m(a')\neq\emptyset$ and a or a' is not in *CONS*(D) (a' for instance), then $m'_i(a)=m(a)\cup m(a')$, $m'_i(a')=\emptyset$, and $m'_i(b)=m(b)$ if $b\neq a,a'$. Otherwise, $m'_i=m$.

else $m'_i=m_i$.

-m_{i+1} is obtained from m'_i as follows: for every tuple $t=a_1...a_p$ such that $m'_i(t)\neq\emptyset$ and $m_i(t)=\emptyset$, add a new integer i_t to $m'_i(a_j)$ for all j in $\{1,...,p\}$.

(4) If m_{i0} is the last function computed at Step 3 then m_0 is obtained by removing from m_{i0} an element from each pair of inseparable integers.

Theorem 2 Algorithm 2 always terminates and if m_0 satisfies the partition constraint then m_0 is a characteristic model of D else D is inconsistent.

Proof

We can notice that for every i, $T(m'_i)=T(m_{i+1})$. So we can use the proof of Theorem 1 in order to conclude that if m_0 does not satisfy the partition constraint, then D is inconsistent and else m_0 is a query model of D. It is also clear that m_0 does not contain any pair of inseparable integers. We show by induction that the following property (H_i) is true for all i.

- (H_i) $\forall t \in T(m_i)$ $\exists k_t \in m_0(t)$ $\forall s \in T(m_0)$ $k_t \in m_i(s) \Rightarrow D \models t \sqsubseteq s$.

(H_1) is easy to test. If (H_i) is true, then let t be a tuple in $T(m_{i+1})$. There are two possibilities (a) t is in $T(m_i)$, (b) t is in $T(m_i')$ but not in $T(m_i)$.

-(a) If t is in $T(m_i)$, let k be the integer associated with t in (Hi). Let X isa A, x and a be the elements used to compute mi+1 from m_i. Let s be a tuple of T(D). If k is in $m_i(s)$ then (H_i) implies $D \models t \sqsubseteq s$. If k is in $m'_i(s)$ but not in $m_i(s)$, then a is a sub-tuple of s (because a is the only value that have changed from m_i to m'_i). We write s=as' and k is necessarily in $m_i(x)$ and in $m_i(s')$ so we have $D \models t \sqsubseteq x$ and $D \models t \sqsubseteq s'$. As we clearly have $D \models x \sqsubseteq a$, we finally obtain $D \models t \sqsubseteq s$.

-(b) If t is in $T(m'_i)$, then we can find a tuple t' in $T(m_i)$ such that $D \models t \sqsubseteq t'$. So, we are in case (a) with t' instead of t.

In conclusion (H_i) is true for all i. We can use the proposition 2 to conclude that m_0 is a characteristic model of the database D.

Example 2 If we consider the database D of Example 1, then Figure 4 shows a characteristic model m_0 of D as computed by Algorithm 2:

m_0: a $\rightarrow$ {1,3}

b $\rightarrow$ {1,2,4} b' $\rightarrow$ {2,6}

c $\rightarrow$ {1,2,4,5} c' $\rightarrow$ {2,6,7} □

Figure 4

We can deduce from m_0 that $D \models b \sqsubseteq c$ (because $m_0(b) \subseteq m_0(c)$) and that $D \not\models a \sqsubseteq b$ (because $m_0(a) \not\subseteq m_0(b)$). It is clear that a minimal model is not enough to characterize the order $\sqsubseteq$. Indeed, with the minimal model m_q of Figure 3, we have $m_q(a) \subseteq m_q(b)$.

Now, let us define formally insertion and deletion of a tuple in a given (simple) database. In doing so, we shall require that the insertion or deletion of a tuple produce "minimal" change in the original database. We consider this to be a reasonable constraint, from a practical viewpoint. Let BASES(Σ) be the set of all consistent databases over the same language L having the same set of dependencies Σ. Recall that we are updating equivalence classes and not particular databases.

Given a class $\bar{D}$ and a tuple t, define the result of the insertion of $\neg t \approx \emptyset$ in $\bar{D}$, denoted INS($\bar{D}, \neg t \approx \emptyset$), to be the class $\bar{D}'$ verifying the following properties:

(1) $\bar{D} \leq \bar{D}'$

(2) $D' \models \neg t \approx \emptyset$

(3) $\bar{D}'$ is a minimal class verifying (1) and (2).

It is clear that the insertion of $\neg t \approx \emptyset$ in $\bar{D}$ does not always produce a consistent database, and, when it does, the result may not be unique. This defines a (partial) function INS from BASES(Σ)/$\equiv$ x TUPLES into BASES(Σ)/$\equiv$. For example, if we insert $\neg ab \approx \emptyset$ in the (class of the) database $D_1 = \{\neg bc \approx \emptyset, B \text{ isa } C\}$, we obtain the (class of the) database $D_2 = \{\neg abc \approx \emptyset, B \text{ isa } C\}$.

We define deletion of a tuple from a class $\bar{D}$, in much the same manner as for insertion. Given a class $\bar{D}$ in BASES(Σ)/$\equiv$ and a tuple t in TUPLES, the deletion of $\neg t \approx \emptyset$

from $\overline{D}$, denoted by DEL($\overline{D}$,$\neg t \approx \emptyset$) is the class $\overline{D'}$ verifying the following properties:

(1) $\overline{D'} \leq \overline{D}$

(2) $D' \not\models \neg s \approx \emptyset$, for all s such that $D \models s \sqsubseteq t$.

(3) $\overline{D'}$ is the maximal class verifying (1) and (2).

We shall see that, as a consequence of Theorem 3 below, DEL($\overline{D}$,$\neg t \approx \emptyset$) is always defined for every value of D or t. For example, if we delete $\neg abc \approx \emptyset$ from the (class of the) database $D_3 = \{\neg abc \approx \emptyset\}$, we obtain the (class of the) database: $D_4 = \{\neg ab \approx \emptyset, \neg bc \approx \emptyset, \neg ac \approx \emptyset\}$. Now if we delete the same formula $\neg abc \approx \emptyset$ from the (class of the) database $D_2 = \{\neg abc \approx \emptyset, B \text{ isa } C\}$ above then we obtain the (class of the) database $D_5 = \{\neg bc \approx \emptyset, \neg ac \approx \emptyset, B \text{ isa } C\}$. Let us note that DEL(INS($\overline{D}$,$\neg t \approx \emptyset$),$\neg t \approx \emptyset$) is generally not equal to D.

We can notice that a characteristic model m_0 of D is also a characteristic model of any database of BASES(Σ)/$\equiv$ equivalent to D. So, given an equivalence class $\overline{D}$ in BASES(Σ)/$\equiv$, we say that m_0 is a characteristic model of D iff m_0 is a characteristic model of all databases in $\overline{D}$. So, in order to process an insertion or a deletion, all we have to do is to compute two mappings i and d such that, given a formula $\neg t \approx \emptyset$, and a caracteristic model m:

i(m,$\neg t \approx \emptyset$) is a caracteristic model of INS($\overline{D}$,$\neg t \approx \emptyset$) and

d(m,$\neg t \approx \emptyset$) is a caracteristic model of DEL($\overline{D}$,$\neg t \approx \emptyset$).

The following algorithm constructs function i.

Algorithm 3 Insertion

Input : A characteristic model m_0 of D.

Output: A characteristic model m'_0 of INS(D,$\neg t \approx \emptyset$) (if it is defined.)

(1) For every sub-tuple $s=a_1 \ldots a_p$ of t, add the integer i_s to $m_0(a_i)$ for i in $\{1,\ldots,p\}$

(2) Compute m_{i+1} from m_i as follows until $m_{i+1}=m_i$.

-if there is X isa A in Σ, and x in dom(X) such that $m_i(x)\neq\emptyset$ and $m_i(a)\cap m_i(x)=\emptyset$, for all a in dom(A).

then m'_i is obtained from m_i by taking a constant a^* in dom(A) such that $m_i(a^*)=\emptyset$ and setting $m'_i(a^*)=m_i(x)$.

else if there is X isa A in Σ, x in dom(X) and a in dom(A) such that $m_i(a)\cap m_i(x)\neq\emptyset$,

then let $m(a)=m_i(a)\cup m_i(x)$ and $m(b)=m_i(b)$ if $b\neq a$. If there is a' in dom(A) such that $m(a)\cap m(a')\neq\emptyset$ and a or a' is not in *CONS*(D) (a' for instance), then $m'_i(a)=m(a)\cup m(a')$, $m'_i(a')=\emptyset$, and $m'_i(b)=m(b)$ if $b\neq a,a'$. Otherwise, $m'_i=m$.

else $m'_i=m_i$.

-m_{i+1} is obtained from m'_i as follows: for every tuple $t=a_1 \ldots a_p$ such that $m'_i(t)\neq\emptyset$ and $m_i(t)=\emptyset$, add a new integer i_t to $m'_i(a_j)$ for all j in $\{1,\ldots,p\}$.

(3) If m_{i0} is the last function computed at step 2, m'_0 is obtained by removing from m_{i0} one element from each pair of inseparable integers.

Theorem 3 Algorithm 3 terminates, and if m'_0 satisfies the partition constraint then $i(m,\neg t\approx\emptyset)=m'_0$ else $INS(\overline{D},\neg t\approx\emptyset)$ is not defined.

proof

The proof of this theorem is quite similar to the proof of Theorem 2.

Algorithm 3 strongly resembles Algorithm 2, and this is not surprising because the construction of a characteristic model for a database D can be seen as the insertion of the tuples of D into the empty database. We give now an algorithm for constructing function d.

Algorithm 4 Deletion

Input: A characteristic model m_0 of D and a tuple t.

Output: A characteristic model m'_0 of $DEL(D, \neg t \approx \emptyset)$.

The model m'_0 is obtained from m_0 as follows:

$m'_0(a) = m_0(a) \setminus m_0(t)$, for every constant a.

Theorem 4 The model m'_0 computed by Algorithm 4 is the characteristic model $d(m, \neg t \approx \emptyset)$.

Proof

We clearly obtain a caracteristic model from Algorithm 4. We can use Proposition 2 to characterise the elements of the input model m_0. We obtain, for any tuple t of T(D),

(1) $m_0(t) = \{k_s \ / \ D \models s \not\subseteq t\}$.

In conclusion, it is necessary and sufficient to remove from m_0 all the integers of $m_0(t)$ in order to eliminate from T(D) the tuples s such that $D \models s \subseteq t$. Property (1) above shows that we do not remove anything else from T(D).

4. CONCLUDING REMARKS

We have seen a proper subset of first order languages that seems to be suitable for data and knowledge representation. In support of this claim, we have presented algorithms for processing queries and updates in a

restricted class, namely simple databases, corresponding to relational databases. Let us note that simple databases are not suitable for modeling recursive queries under the partition constraint. Indeed this constraint implies that, for any constants a, b and c of the same type, the formulas $\neg ab \approx \emptyset$ and $\neg bc \approx \emptyset$ always lead to inconsistent databases. However, if we move to a larger class of databases, namely by replacing the partition constraint by the inclusion or the transitive constraint mentioned earlier, then recursion is possible. We are currently working in this direction.

BIBLIOGRAPHY

[C70] E.F.Codd,"A Relational Model of Data for Large Shared Data Banks", CACM 13:6, june 1970, p.377-387.

[CKS85] S. Cosmadakis, P. Kanellakis, N. Spyratos, "Partition Semantics For Relations", Proceedings ACM PODS, March 1985 (also to appear in JCSS).

[F82] R. Fagin,"Horn clauses and database dependencies", J.ACM 29,4, Oct. 1982, p.952-985.

[FUV83] R. Fagin, J.D. Ullman, M.Y. Vardi,"On the Semantics of Updates in Databases", Proceedings ACM PODS, 1983, p.352-365.

[FV84] R. Fagin, M.Y. Vardi,"The theory of data dependencies-a survey", IBM Research Report RJ 4321, june 1984.

[L66] R.C. Lyndon,"Notes in Logic", Van Nostrand Mathematical Studies,1966.

[R83] R. Reiter,"Towards a logical reconstruction of relational database theory", in : On Conceptual Modeling: Perspective from Artificial Intelligence, Databases, and Programming languages.

[S84] N. Spyratos,"The Partition Model: a Deductive Database Model",INRIA Research Report No 286, April 1984 (also to appear in ACM TODS).

[SL86] N. Spyratos, C. Lécluse, "The Semantics of Queries and Updates in Relational Databases",INRIA Research Report No 561, June 1986.

[SL87] N. Spyratos, C. Lecluse. "Incorporating Fonctional dependencies in deductive query answering", IEEE International conference on Data Ingeneering, Los Angeles, February 1987.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249-6399