PARTITION SEMANTICS FOR INCOMPLETE INFORMATION IN RELATIONAL DATABASES

D. LAURENT
LIFO, University of Orléans
BP 6759, 45067 Orléans Cedex 2
France
uucp: mcvax!inria!univorl!laurent

N. SPYRATOS
LRI, University of Paris-Sud
91405 Orsay
France

ABSTRACT

We define partition semantics for databases with incomplete information and we present an algorithm for query processing in the presence of incomplete information and functional dependencies. We show that Lipski's model for databases with incomplete information can be seen as a special case of our model.

1. INTRODUCTION

In the relational model of data one views the database as a collection of relations, where each relation is a set of tuples over some domains of values. One notable feature of the relational model is the absence of semantics: a tuple in a relation represents a relationship between certain values, but from the mere syntactic definition of the relation, we know nothing about the nature of the relationship.

One approach in order to remedy this deficiency of the relational model is proposed in [11], where tuples are seen as strings of uninterpreted symbols. Interpretations for these symbols are provided using subsets of an underlying population of objects, and set containment provides the basic inference mechanism. Let us explain the approach informally, using an example.

Consider the following (very small) database, containing only two tuples

AGE SITUATION	SITUATION SEX
young unemployed	*unemployed female*

The tuple *young unemployed* can be seen as a string of two uninterpreted symbols, *young* and *unemployed*. Now, think of a possible world, and let W be the set of all individuals in that world. Moreover, let *I(young)* be the set of all individuals of W that are young, and let *I(unemployed)* be the set of all individuals of W that are unemployed. Call *I(young)* the **interpretation of the symbol** *young*, and *I(unemployed)* the interpretation of the symbol *unemployed*. Clearly, the intersection *I(young)* $\cap$ *I(unemployed)* is the set of all individuals of W that are both young and unemployed. It is precisely this intersection that we define to be the interpretation of the tuple *young unemployed*. That is,

I(young unemployed) = *I(young)* $\cap$ *I(unemployed)*.

In other words, the interpretation of a tuple is the **intersection** of the interpretations of its constituent symbols.

This kind of set-theoretic semantics for tuples allows for a very intuitive notion of truth. Namely, a tuple t is called *true* in I iff *I(t)* is nonempty. Thus, for example, the (atomic) tuple *young* is true iff *I(young)* is nonempty, that is, iff there is at least one individual in W which is young. Similarly, the tuple *unemployed* is true in I iff there is an individual in W which is unemployed. Finally, the tuple *young unemployed* is true in I iff there is an individual in W which is both young and unemployed.

The set-theoretic semantics just introduced allows for a very natural notion of inference trough set-containment. To see this, consider the following question:

Assuming that

(1) *young unemployed* is true in I

and that

(2) *unemployed female* is true in I

can we infer that

(3) *young unemployed female* is true in I.

If we recall the definition of truth given earlier, then we can reformulate this question as follows:

Assuming that

(1') $I(young) \cap I(unemployed) \neq \phi$

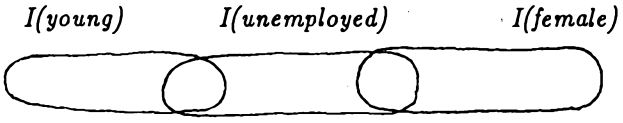
and that

(2') $I(unemployed) \cap I(female) \neq \phi$

can we infer that

(3') $I(young) \cap I(unemployed) \cap I(female) \neq \phi$.

Clearly, the answer depends on the interpretation *I*, and the following diagram shows a configuration where the answer is **no**.



However, if we impose **constraints** on the interpretation *I*, for instance, if we require that

(4') $I(unemployed) \subseteq I(young)$

or

(5') $I(unemployed) \subseteq I(female)$

then the answer is **yes**. Roughly speaking, we can summarize our example as follows:

[(1') and (2')] does *not* imply (3')

whereas

[(1') and (2')] and [(4') or (5')] implies (3').

It turns out that constraints such as (4') and (5') provide the right interpretations for functional dependencies and other semantic constraints (see [11]). Notice that (4') and (5') can be put into words as follows:

(4) all unemployed individuals are young

(5) all unemployed individuals are female.

Thus, roughly speaking we can say that

[(1) and (2)] does *not* imply (3)

whereas

[(1) and (2)] and [(4) or (5)] implies (3).

We have seen so far the basic ingredients of the model introduced in [11], known as the *partition model*. The partition model has been used to study variants of the weak instance assumption [1], as well as to derive algorithms for (deductive) query and update processing [6, 12, 13]. In this paper we extend the partition model in order to provide semantics for databases with incomplete information.

The paper is organized as follows. In Section 2 we give formal definitions of the following concepts: database, interpretation, truth, model and semantic implication. In Section 3, we study incomplete information by considering two kinds of tuples: sure and possible tuples; then, we restrict our attention to a specific case (that subsumes Lipski's model of incomplete information [7, 8]) and we give computational algorithms. Section 4 contains some concluding remarks and suggestions for further work. Throughout the paper, we assume some familiarity with the basic relational terminology.

2. THE MODEL

We shall consider separately the syntax and the semantics of our model. The syntactic part is essentially the relational model. The semantic part is a formalization of the concepts already explained in the introduction.

2.1. Syntax

We begin with a finite, nonempty set $U = \{A_1,...,A_n\}$. The set U is called the *universe* and the A_i's are called the *attributes*. Each attribute A_i is associated with a countably infinite set of symbols (or values) called the *domain* of A_i and denoted by $dom(A_i)$.

We assume that $U \cap dom(A_i)$ is empty for all i, and that $dom(A_i) \cap dom(A_j) = \phi$ for $i \neq j$ (we shall discuss the consequences of this last assumption in Section 4). A *relation scheme* over U is a nonempty subset of U; we call sch(U) the set of all relation schemes over U and a relation scheme is denoted by the juxtaposition of its attributes (in any order).

A *tuple* t over a relation scheme R is a function defined on R such that $t(A_i)$ is in $dom(A_i)$, for all A_i in R. We denote by dom(R) the set of all tuples over R. Clearly dom(R) is the cartesian product of the domains of all the attributes in R. If t is a tuple over $R = A_1...A_n$, and if $t(A_j) = a_j$, $j = 1,...,n$, then we denote the tuple t as $a_1...a_n$. A *relation* over R is a set of tuples over R. Thus a relation over R is a subset of dom(R).

Definition 2.1 A *database* over U is a pair $D = (\delta,\Sigma)$ such that

(1) δ is a function assigning to *every* relation scheme R a *finite* relation over R, and

(2) Σ is a set of ordered pairs (X,Y) such that X and Y are subsets of U.

Every pair (X,Y) in Σ is called a *functional dependency* and is denoted as $X \rightarrow Y$. □

Example 2.1 Consider a universe of three attributes, say U = {A,B,C}, with the following associated domains

$dom(A) = \{a_1,a_2,...\}$, $dom(B) = \{b_1,b_2,...\}$,
$dom(C) = \{c_1,c_2,...\}$.

Define a function δ on relation schemes over U as follows:

$\delta(AB) = \{a_1b_1, a_2b_1\}$
$\delta(BC) = \{b_1c_1, b_1c_2\}$
$\delta(R) = \phi$ for all R different than AB and BC.

Let Σ be the set $\{A \rightarrow B, BC \rightarrow A\}$. The database (δ,Σ) just defined is shown in Figure 2.1(b). The function δ is represented by two tables, with schemes AB and BC as headers and the corresponding tuples as rows. Clearly, the convention used here is that relation schemes that are assigned empty relations are not represented. □

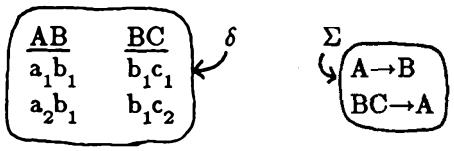
$U = \{A,B,C\}$, $dom(A) = \{a_1,a_2,...\}$,
$dom(B) = \{b_1,b_2,...\}$, $dom(C) = \{c_1,c_2,...\}$

(a)

I

$a_1 \rightarrow \{1,3\}$ $\quad a_2 \rightarrow \{2,4\}$
$b_1 \rightarrow \{1,2,3,4\}$ $\quad b_2 \rightarrow \{5,6\}$
$c_1 \rightarrow \{3\}$ $\quad c_2 \rightarrow \{1\}$
$x \rightarrow \phi$, for every x different than $a_1, a_2, b_1, b_2, c_1, c_2$.

(b)



(c)

$I(a_1b_1) = I(a_1) \cap I(b_1) = \{1,3\}$
$I(a_2b_1) = I(a_2) \cap I(b_1) = \{2,4\}$
$I(b_1c_1) = I(b_1) \cap I(c_1) = \{3\}$
$I(b_1c_2) = I(b_1) \cap I(c_2) = \{1\}$
$I(a_1b_1c_1) = I(a_1) \cap I(b_1) \cap I(c_1) = \{3\}$
$I(a_1b_1c_2) = I(a_1) \cap I(b_1) \cap I(c_2) = \{1\}$

FIGURE 2.1

Given a database D = (δ,Σ), call *database scheme* the set of all relation schemes that are assigned non-empty relations under δ. That is, the relation scheme of D is the set

$$\{R \in sch(U) \mid \delta(R) \neq \phi\}.$$

Thus, the database scheme in Example 2.1 consists of the relation schemes AB and BC.

2.2. Semantics

We assume that the "real world" consists of a countably infinite set of objects, and we identify these objects with the positive integers. Let ω be the set of all positive integers, and let $2^\omega = \{\tau \mid \tau \subseteq \omega\}$ be the set of all subsets of ω. The set 2^ω is the semantic domain in which tuples and dependencies receive their interpretation.

Throughout our discussions, we consider fixed the universe of attributes $U = \{A_1,A_2,...,A_n\}$, and the associated domains $dom(A_i)$. For notational convenience, we denote by SYMBOLS the union of all attribute domains, and by TUPLES the union of all domains. That is:

$$SYMBOLS = \cup_{A \in U} dom(A),$$
$$TUPLES = \cup_{R \in sch(U)} dom(R).$$

Clearly, SYMBOLS is a subset of TUPLES.

Definition 2.2 An *interpretation* of U is a function I from SYMBOLS into 2^ω such that

$$\forall A \in U, \forall a, a' \in dom(A),$$
$$[\, a \neq a' \Rightarrow I(a) \cap I(a') = \phi \,].$$ □

Thus the basic property of an interpretation is that *different* symbols of the *same* domain are assigned disjoint sets of integers. In Figure 2.1(a) we see a function I satisfying this property. The intuitive motivation behind this definition is that an attribute value, say *a*, is a (atomic) property, and I(a) is a set of objects having property *a*. Furthermore, an object cannot have two different properties *a*, *a'* of the same "type"; hence $I(a) \cap I(a') = \phi$ (we shall discuss the consequences of this restriction in Section 4).

Given an interpretation I, we extend it from SYMBOLS to TUPLES as follows:

$$\forall R \in sch(U), \forall a_1a_2...a_k \in dom(R),$$
$$I(a_1a_2...a_k) = I(a_1) \cap ... \cap I(a_k).$$

In Figure 2.1(c) we see some examples of computations. The intuitive motivation for this extension is that a tuple, say ab, is the conjunction of the (atomic) properties *a* and *b*. Accordingly, I(ab) is the set of objects having both properties *a* and *b*; hence we have $I(ab) = I(a) \cap I(b)$. Notice that the basic property of an interpretation is satisfied by the extension, namely:

$$\forall R \in sch(U), \forall t, t' \in dom(R),$$
$$[\, t \neq t' \Rightarrow I(t) \cap I(t') = \phi \,].$$

Our definition of an interpretation suggests an intuitive notion of truth: a property (tuple) t is true in an interpretation I, if there is at least one object having property t under I.

Definition 2.3 Let D = (δ,Σ) be a database over U. An interpretation I of U is called a *model* of D if

(1) $\forall R \in sch(U), \forall t \in \delta(R), I(t) \neq \phi$

(2) $\forall X \rightarrow Y \in \Sigma, \forall x \in dom(X), \forall y \in dom(Y),$
$[\, I(x) \cap I(y) \neq \phi \Rightarrow I(x) \subseteq I(y) \,]$. □

Roughly speaking, we say that I is a model of D if

(1) I verifies every tuple t in δ (in the sense $I(t) \neq \phi$), and

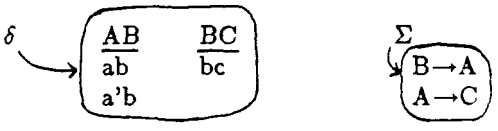
(2) I satisfies every functional dependency X→Y in Σ (in the sense that the set $\{(x,y) \mid x \in dom(X), y \in dom(Y), I(x) \cap I(y) \neq \phi\}$ is a function).

In Figure 2.1 we see an interpretation I and a database (δ,Σ). Using I and the results of the computations of Figure 2.1(c) we can verify that I is a model of (δ,Σ), as follows:

(1) The tuples appearing in the database are a_1b_1, a_2b_1, b_1c_1, b_1c_2, and they all receive non-empty interpretations under I.

(2) For A→B: the only true tuples over AB with respect to I are a_1b_1, a_2b_1, and we have $I(a_1) \subseteq I(b_1)$ and $I(a_2) \subseteq I(b_1)$. Thus A→B is satisfied. For BC→A: the only true tuples over ABC with respect to I are $a_1b_1c_1$ and $a_1b_1c_2$, and we have $I(b_1c_1) \subseteq I(a_1)$ and $I(b_1c_2) \subseteq I(a_1)$. Thus BC→A is also satisfied. □

Having defined the concepts of interpretation, truth, and model, we can now define the concept of consistency. A database D is called *consistent* iff D possesses at least one model; and otherwise D is called *inconsistent*. For example, the database D = (δ,Σ) of Figure 2.1 is consistent as the interpretation shown in that same figure is a model of D. On the other hand, the following database is inconsistent, as no interpretation I can verify the tuples ab and ab', and the dependency B→A, all at the same time.



Indeed, assume that there is a model, say I. Then, Definition 2.3(1) implies that

$I(a) \cap I(b) \neq \phi$ and $I(a') \cap I(b) \neq \phi$

and, using Definition 2.3(2), we obtain (because of B→A)

$I(b) \subseteq I(a)$ and $I(b) \subseteq I(a')$.

It follows that $I(b) \subseteq I(a) \cap I(a')$ and, as $I(b) \neq \phi$ (this follows from $I(a) \cap I(b) \neq \phi$) we obtain that

$I(a) \cap I(a') \neq \phi$.

On the other hand, as a and a' belong to the same domain, it follows, from Definition 2.2, that $I(a) \cap I(a') = \phi$, a contradiction. Thus the database possesses no model and, therefore, it is inconsistent. In our discussions, for the remaining of this paper, we shall always assume that databases are consistent. □

Now, given a (consistent) database D and a model I of D, all tuples appearing in the database are true in I (by definition). However, it may happen that some tuples that do not appear explicitly in the database are also true in I. For example, in the database of Figure 2.1, tuples $a_1b_1c_1$, $a_1b_1c_2$ fall into this category, as $I(a_1b_1c_1) \neq \phi$ and $I(a_1b_1c_2) \neq \phi$. (Note, however, that tuples $a_2b_1c_1$, $a_2b_1c_2$ are not true in that interpretation.)

Definition 2.4 Let D = (δ,Σ) be a database over universe U. Let t be any tuple in TUPLES. We say that D *implies* t, denoted $D \models t$, if t is true in every model of D. □

Clearly, D implies all tuples appearing in D. For example, in Figure 2.1, $D \models a_1b_1$, a_2b_1, b_1c_1, b_1c_2. This follows immediately from the definition of a model. As a consequence, D also implies the symbols a_i, b_i, c_i for i =1, 2. Indeed, consider for instance a_1. In every model m of D, we have:

$m(a_1b_1) = m(a_1) \cap m(b_1) \neq \phi$.

Thus, $m(a_1)$ is not empty for any model m of D; that is, by Definition 2.4, $D \models a_1$.

On the other hand, D does not imply the tuples $a_2b_1c_1$, $a_2b_1c_2$, as they are false in the model I shown in Figure 2.1.

Notice that D does not imply the tuples $a_1b_1c_1$, $a_1b_1c_2$ either. Indeed, although these tuples are true in the model I of Figure 2.1, we can find a model I' of D in which these tuples are false. Indeed, define I' such that $I'(a_1) = \{2,4\}$, and $I'(a_2) = \{1,3\}$, and $I'(x) = I(x)$ for all $x \neq a_1, a_2$. Then I' is a model of D falsifying $a_1b_1c_1$, $a_1b_1c_2$ (and, moreover, verifying

$a_2b_1c_1$, $a_2b_1c_2$). □

Given a model m of D, we denote by T(m) the set of all tuples in TUPLES which are true in m. Let us denote with mod(D) the set of all models of D, then we define

$$T(D) = \cap_{m \in mod(D)} T(m).$$

Clearly, T(D) is the set of all tuples which are true in *every* model of D, hence

$$\forall t \in TUPLES,\ D \models t \iff t \in T(D).$$

3. INCOMPLETE INFORMATION

3.1. Information Systems

In many applications the information to be recorded in the database may be uncertain. For example, we may know that:

"John is a *clerk* or a *driver"*

without knowing exactly what John's job is. Of course, we will have to record both possibilities in our database, and we will have to take them into account when processing queries and updates. Conceptually, having to deal with these two possibilities is tantamount to having to deal with two possible databases, one containing the information *"John* is a *clerk"*, and the other containing the information *"John* is a *driver"*. As the two databases refer to the same application they are likely to have the *same* set of dependencies. However, in general, there may be uncertainty not only at the level of tuples, as in our example, but also at the level of dependencies as well. Leaving aside the general case, we shall concentrate to sets of databases having the same set of dependencies.

Definition 3.1 Let U be a universe. An *information system* over U is a finite set of consistent databases over U, having the same set of dependencies. □

In Figure 3.1 we see an information system consisting of two databases, $D_1 = (\delta_1, \Sigma)$ and $D_2 = (\delta_2, \Sigma)$.

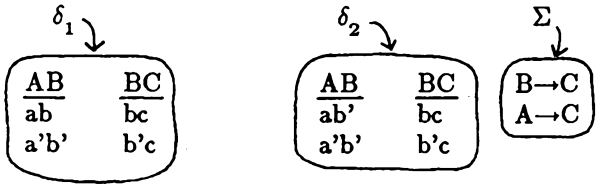


FIGURE 3.1

It is important to note that our information systems generalize the information systems proposed and studied by Lipski in [7, 8]. Indeed, Lipski's systems can be viewed as satisfying the following properties.

(1) All databases in the system have the same scheme which contains a single relation scheme, namely, the universe U itself.

(2) There is a distinguished attribute O (representing the set of objects in the system) in U, and all functional dependencies in Σ are of the form O→A, where A is an attribute.

(3) If the system contains the tuples oat and oa't' then it must also contain the tuples oa't and oat'.

It is easy to see that these properties define an information system in the sense of Definition 3.1. Now, given an information system IS we shall be interested in tuples that either are implied by every database of IS, or are implied by at least one database of IS.

Definition 3.2 Let IS be an information system over a universe U. Let t be a tuple in TUPLES. Then

t is called *sure* iff $D \models t$, for all D in IS

t is called *possible* iff $D \models t$, for some D in IS. □

For example, in the information system of Figure 3.1, the tuple a'b'c is a sure tuple as it is implied by D_1 and by D_2 (this is so because of the dependency B→C). Similarly, the tuple ac is a sure tuple. Indeed, D_1 implies abc because of the dependency B→C, and therefore, D_1 implies ac. Similarly, ab'c and thus ac are implied by D_2.

On the other hand, the tuple abc is not sure, as it is implied by D_1 but not by D_2; nevertheless, this tuple is possible as it is implied by D_1.

Finally, let us consider a", a symbol in dom(A) different than a and a'. The tuple a"c is not possible (and therefore not sure either), as it is not implied neither by D_1 nor by D_2. □

Given an information system IS = $\{D_1, D_2, ..., D_n\}$, let $T_S(IS)$ denote the set of all sure tuples of IS, and let $T_P(IS)$ denote the set of all possible tuples of IS. Clearly, $T_S(IS) \subseteq T_P(IS)$.

Now, recall from the previous section, that T(D) denotes the set of all tuples implied by D (where D is a given database). It follows from Definition 3.2 that:

$$T_S(IS) = T(D_1) \cap T(D_2) \cap ... \cap T(D_n)$$

$$T_P(IS) = T(D_1) \cup T(D_2) \cup ... \cup T(D_n).$$

Clearly, in order to process queries in an information system IS, we must be able to compute the sets $T_S(IS)$ and $T_P(IS)$. To do this, we need the concept of a model M for an information system. In order to understand this concept, consider an information system IS = $\{D_1, D_2, ..., D_n\}$, and let m_1, m_2, ..., m_n be n models, one for each database in the system. Now, for each symbol x in SYMBOLS, define

$$M_i(x) = \{(k,i) \mid k \in m_i(x)\},$$

for every i in {1,2,...,n} and, using $M_i(x)$, define M(x) as follows:

$$M(x) = M_1(x) \cup M_2(x) \cup ... \cup M_n(x).$$

The model thus defined is extended to the set TUPLES as in Section 2: if t is the tuple $a_1a_2...a_k$, then

$$M(a_1a_2...a_k) = M(a_1) \cap M(a_2) \cap ... \cap M(a_k).$$

For example, in order to construct a model M for the system of Figure 3.1, let us consider two models m_1 and m_2, one for each database in the system:

m_1

a→{1} a'→{2,3} a"→{4}
b→{1,2} b'→{3} c→{1,2,3,4}
x→ϕ for every x different than a, a', a", b, b', c.

m_2

a→{1,4} a'→{2,3} a"→ϕ
b→{2,4} b'→{1,3} c→{1,2,3,4}
x→ϕ for every x different than a, a', a", b, b', c.

Now, we define the functions M_1 and M_2 and we construct M.

M_1

a→{(1,1)} a'→{(2,1),(3,1)}
a"→{(4,1)} b→{(1,1),(2,1)}
b'→{(3,1)} c→{(1,1),(2,1),(3,1),(4,1)}
x→ϕ for every x different than a, a', a", b, b', c.

M_2

a→{(1,2),(4,2)} a'→{(2,2),(3,2)}
a"→ϕ b→{(2,2),(4,2)}
b'→{(1,2),(3,2)} c→{(1,2),(2,2),(3,2),(4,2)}
x→ϕ for every x different than a, a', a", b, b', c.

M

a→{(1,1),(1,2),(4,2)} a'→{(2,1),(3,1),(2,2),(3,2)}
a"→{(4,1)} b→{(1,1),(2,1),(2,2),(4,2)}
b'→{(3,1),(1,2),(3,2)}
c→{(1,1),(2,1),(3,1),(4,1),(1,2),(2,2),(3,2),(4,2)}
x→ϕ for every x different than a, a', a", b, b', c.

In the model just defined the interpretation of the tuple a'b'c is computed as follows:

$$M(a'b'c) = M(a') \cap M(b') \cap M(c) = \{(3,1),(3,2)\}.$$

Similarly, for the tuple a"c, we have:

$$M(a"c) = M(a") \cap M(c) = \{(4,1)\}.$$ □

So a model for an information system can be seen as a way to deal with a model of each database in the system without having to *explicitly* store all of them. This concept of a model for an information system is used in the computation of the sets $T_S(IS)$ and $T_P(IS)$ because it allows us to take into account the information present in all databases without having to examine all of them separately. Our approach is the following: first, we compute a special model of IS incorporating the information necessary for the computation of the sets $T_S(IS)$ and $T_P(IS)$, then we proceed to the actual computation.
In what follows we give the details of the computational algorithms.

3.2. Algorithms

First, let us recall that all databases of an information system have the same set of dependencies, say Σ. Let us also recall that, given a set of dependencies Σ we can always find an equivalent set Σ' such that each dependency in Σ' has a single attribute on the right-hand side (see [9] for details). Thus, without loss of generality, we shall assume that each dependency of Σ has a single attribute on the right-hand side. Under this assumption, we present now our main algorithm.

ALGORITHM 1

INPUT : An information system IS = $\{D_1, D_2, ..., D_n\}$
OUTPUT : A model of IS

Initialization

Step 1 With every distinct tuple t in IS associate a distinct positive integer i_t.

Step 2 With every distinct tuple t in IS associate the set of pairs $\{(i_t,k) \mid t \text{ is in } D_k\}$.

Step 3 For every symbol x of SYMBOLS define
$M^0(x) = \{(i_t,k) \mid t \text{ contains } x\}$.

Closure

Step 4 For $j \geq 0$ compute M^{j+1} from M^j as follows:
if there is X→A in Σ, and x in dom(X), and
a in dom(A) such that $M^j(x) \cap M^j(a) \neq \phi$

then $M^{j+1}(a) = M^j(a) \cup$
$\{(i,k) | (i,k) \in M^j(x), k \in \pi_2(M^j(x) \cap M^j(a))\}$

and for all a' different than a,
$M^{j+1}(a') = M^j(a')$

else $M^{j+1} = M^j$

(*Note:* π_2 denotes the projection of the pairs from $\omega \times \{1,2,...,n\}$ over their second component.)

Step 5 Set $M_q = M^l$, where l is the least integer such that $M^{l+1} = M^l$. □

Let us apply ALGORITHM 1 to the system of Figure 3.1.

Step 1 The tuples in the system are ab, ab', a'b', bc and b'c. We number them as follows:

ab→1, ab'→2, a'b'→3, bc→4, b'c→5.

Step 2 We now associate each tuple with the appropriate set of pairs:

ab→{(1,1)} ab'→{(2,2)}
a'b'→{(3,1),(3,2)} bc→{(4,1),(4,2)}
b'c→{(5,1),(5,2)}

Step 3 Then, M^0 is defined as follows:

a→{(1,1),(2,2)} a'→{(3,1),(3,2)}
b→{(1,1),(4,1),(4,2)} b'→{(2,2),(3,1),(3,2),(5,1),(5,2)}
c→{(4,1),(4,2),(5,1),(5,2)}
x→ϕ for every x different than a, a', b, b', c.

Continuing with Steps 4 and 5, we obtain the following model M_q of IS:

a→{(1,1),(2,2)} a'→{(3,1),(3,2)}
b→{(1,1),(4,1),(4,2)} b'→{(2,2),(3,1),(3,2),(5,1),(5,2)}
c→{(1,1),(2,2),(3,1),(3,2),(4,1),(4,2),(5,1),(5,2)}
x→ϕ for every x different than a, a', b, b', c. □

The following theorem guarantees that ALGORITHM 1 terminates and states formally how the sets $T_S(IS)$ and $T_P(IS)$ are computed, thus summarizing the main results of this paper.

Theorem 3.1 ALGORITHM 1 terminates and we have:

$t \in T_S(IS)$ iff $\pi_2(M_q(t)) = \{1,2,...,n\}$

$t \in T_P(IS)$ iff $M_q(t) \neq \phi$. □

Proof

We give here only a sketch of the proof because of a lack of place. The interested reader will find the complete proof in [5].

ALGORITHM 1 terminates because there is a *finite* number of tuples recorded in an information system, and thus Step 1 defines a finite number of pairs.
For all k in {1,2,...,n}, let m_k be the function from SYMBOLS into 2^ω defined as follows:

$\forall x \in$ SYMBOLS, $m_k(x) = \{i \mid (i,k) \in M_q(x)\}$.

In the same way as in [12], we show by induction on the integer j of Step 4 that the m_k's are models of the databases D_k of IS and moreover that these models characterize the sets $T(D_k)$; that is, for all k in {1,2,...,n}, we have:

$\forall t \in$ TUPLES, $t \in T(D_k) \Leftrightarrow m_k(t) \neq \phi$.

Thus M_q is a model of IS and the theorem follows from this result and from Definition 3.2. □

Referring to Figure 3.1, we can apply Theorem 3.1 to compute the sets $T_S(IS)$ and $T_P(IS)$. We obtain, as said in Section 3.1, that a'b'c and ac are in $T_S(IS)$ because we have:

$M_q(a'b'c) = \{(3,1),(3,2)\}$ and

$M_q(ac) = \{(1,1),(2,2)\}$

and thus:

$\pi_2(M_q(a'b'c)) = \pi_2(M_q(ac)) = \{1,2\}$.

Similarly, abc is not in $T_S(IS)$. Indeed,

$M_q(abc) = \{(1,1)\}$ so $\pi_2(M_q(abc)) = \{1\}$.

But this tuple belongs to $T_P(IS)$ because $M_q(abc)$ is not empty.

Finally, a"c is not in $T_P(IS)$ because $M_q(a")$ and thus $M_q(a"c)$ are empty. □

4. CONCLUDING REMARKS

We have seen a set-theoretic interpretation of the relational model, that allows for a natural interpretation of truth and inference in a common framework. We have then argued that sets of databases with common set of dependencies describe an important aspect of incomplete information. We have called such sets information systems, and we have shown that they include those introduced by Lipski as a special case. Finally, we have given algorithms for computing sure tuples (i.e. tuples implied by every database in the system), and possible tuples (i.e. tuples that are implied by at least one database in the system).

So, in this paper, we have studied query processing in information systems for queries such as: "Give all tuples over scheme R that are sure (or possible) in the information system". Selection conditions as

defined in [4] can also be treated in the present framework (see [5]) as follows: we use the particular model that we have considered here in order to "translate" the condition of the selection into a set of sets of pairs; then we compute the sure and possible answers to the query in the same way as shown in Theorem 3.1.

One must notice that our method is correct in the sens that it avoids the problems of soundness and completeness that are dealt with in [7, 8, 10]. Moreover, the translation process of the conditions of selection may be compared with the "Two Phase Query Processing" considered in [2].

First of all, that approach allows for computing only what we called here the *sure* answers to queries. Moreover, the translation process developed in [2] refers to databases in which the information is recorded following a *fixed* scheme, which is not the case in our model. So, it turns out that we have considered a more general case than in [2], but an important aspect of our approach that we have to study is the update problem.

Indeed, while in [2] updates do not alter the calculated expression of a given query, such is not the case in our approach, because, for any modification in the information system, we must in fact update the model of the system.

However, the set-theoretic semantics that we have used seems to be adapted for the processing of updates because this problem is treated in [6, 13] for databases without incomplete information.

Now, let us recall that in the set-theoretic interpretation that we have used, we have required that attribute values in the same domain be interpreted by disjoint sets. This requirement which allows for a natural interpretation of functional dependencies, has an important consequence on the expressive power of the model.

Namely, our set-theoretic semantics is not adequate in the case where two different attributes have the *same* domain. Indeed, if a tuple ab is in the system, where a and b belong to the *same* domain, then we must have, at the same time, $I(a) \cap I(b) = \phi$, because a and b are in the same domain *and* $I(a) \cap I(b) \neq \phi$, because ab is in the system (and, therefore, ab is true in *every* model). We are currently investigating appropriate extensions of our model, in order to deal with this difficulty.

REFERENCES

[1] S. S. Cosmadakis, P. C. Kanellakis, N. Spyratos, "Partition Semantics for Relations", Proc. ACM PODS 1985, *JCSS*, 33-2, 1986.

[2] T. Imielinski, "Query Processing in Deductive Databases with Incomplete Information", In ACM SIGMOD, 1986.

[3] D. Laurent, "Information Incomplète Explicite dans le Modèle Partitionnel de Bases de Données", In 2^e Journées Bases de Données Avancées, Giens (France), *INRIA Ed.*, 1986.

[4] D. Laurent, "La Logique des Partitions : Application à l'Information Disjonctive dans les Bases de Données", Thèse de troisième cycle, University of Orléans, Jan. 1987.

[5] D. Laurent, N. Spyratos, "Partition Semantics for Query-Answering in Relational Databases with Incomplete Information", rep. LIFO n°87-6, University of Orléans.

[6] Ch. Lecluse, "Une Sémantique Ensembliste pour les Bases de Données, Application au Modèle Relationnel", Thèse de troisième cycle, University of Paris-Sud, March 1987.

[7] W. Lipski Jr, "On Semantic Issues Connected with Incomplete Information Databases", *ACM TODS*, 4-3, Sept. 1979.

[8] W. Lipski Jr, "On Databases with Incomplete Information", *JACM*, 28, pp 41-70, 1981.

[9] D. Maier, "The Theory of Relational Databases", *Pitman*, 1983.

[10] R. Reiter, "A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values", *JACM*, 33-2, 1986.

[11] N. Spyratos, "The Partition Model: a Deductive Database Model", *ACM TODS*, March 1987.

[12] N. Spyratos, Ch. Lecluse, "Incorporating Functional Dependencies in Deductive Query Answering", Proc. International Conference on Data Engeneering, Los Angeles, Feb. 1987.

[13] N. Spyratos, Ch. Lecluse, "The Semantics of Queries and Updates in Relational Databases", rep. INRIA n°561, Aug. 1986.