The semantics of queries and updates in relational databases

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 Cedex
France
Tél.(1) 39 63 55 11

Rapports de Recherche

N° 561

THE SEMANTICS OF QUERIES AND UPDATES IN RELATIONAL DATABASES

Christophe LECLUSE
Nicolas SPYRATOS

Août 1986

THE SEMANTICS OF QUERIES AND UPDATES IN RELATIONAL DATABASES

Christophe LECLUSE
Nicolas SPYRATOS

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

ABSBRACT : We present a set-theoretic interpretation of the relational model which allows a semantic approach to query and update processing. Our model is deductive in the sense that new tuples can be deduced from those that were explicitly introduced in the database. We give algorithms for answering queries and performing updates, and for determining equivalence between databases.

RESUME : Nous présentons une interprétation ensembliste du modèle relationnel permettant une approche sémantique du traitement des requêtes et des mises-à-jour. Notre modèle est déductif en ce sens que de nouveaux n-uplets peuvent être déduits à partir des n-uplets explicitement dans la base. Nous présentons des algorithmes (a) pour répondre aux requêtes (b) pour effectuer des mises-à-jour, et (c) pour vérifier si deux bases de données sont équivalentes.

25 Juin 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 [2]. These attempts have concentrated, mainly, around the following issues:

Semantic constructs: In the relational model, one views the database as a collection of relations, where each relation is a set of tuples over some domains of values. However, one notable feature of the relational model is the complete 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. Semantic information, such as functional dependencies, may be used to specify which databases are meaningful and which are meaningless. However, the specification of such information is done mostly by means external to the relational model, usually first order logic [9] or, so-called semantic networks [4]. Thus the use of semantic information in the processing of relational queries and updates becomes very difficult-if not downright impossible. ▫

Deductive capability: A significant part of the effort to improve the power of database systems has been in development of deductive databases. As defined in [6], "a deductive database is a database in which new facts may be derived from facts that were explicitly introduced". A very important difference between a deductive and a conventional database is that, in the former, new facts may be derived recursively. This very characteristic of the deductive databases is what makes query processing a difficult task in such an environment. ▫

Universal relation interface: In the relational model, a query is a well-formed expression whose operands are relation schemes over U, and whose operations are projection, selection, join, and set theoretic operations [12]. In order to compute the answer to a query, we substitute database relations for corresponding schemes in the expression, and we perform the operations. The basic problem with this definition is that the user is required to know how attributes are grouped together into relation schemes. To avoid this problem, there are now a number of attempts to use a universal relation as the basis of a query language, for example System U [11]. In such a language, although the actual database is a multi-relation database, the user is aware *only* of the universe of attributes. This universe is the interface through which the user interacts with the database as follows:

Queries: The user submits a set Q of attributes and a selection condition and the system returns the set of tuples over Q "implied" by the database and

satisfying the selection condition.

Updates: The user submits the tuple(s) to be inserted or deleted and the system returns an acknowledgement informing the user whether the update is acceptable (and therefore performed).

However, ambiguities arise, as the system must "navigate" through relation schemes in order to answer queries or to perform updates. These ambiguities come from the purely *syntactic* approach to query answering and update processing in the relational model.

One approach to remedy these defficiencies of the relational model, 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, as in [ ]. These interpretations either verify or falsify the schemes and constraints. An interesting body of theory was developed around this approach, particularly with respect to special types of sentences about relations called dependencies (see [10] for survey of the area).

Here we adopt a different approach, first proposed in [10], and known as *the partition model*. In this approach, the relation schemes, the constraints *and* the database are seen as strings of uninterpreted symbols. Interpretations for these symbols are provided using subsets of an underlying population of objects. Deduction of new tuples is done essentially using set-containment. Semantic information, such as functional dependencies or ISA-relationships, are seen as restrictions on interpretations and can be readily incorporated in the deductive process. This approach, while keeping the syntactic simplicity of the relational model, and the philosophy of the universal relation interface, adds to it a deductive component through set-containment.

In this paper we study the semantics of query and updates in relational databases. The approach that we adopt is a recasting of the partition model suitable for our goals. Let us see an example introducing (informally) the problems that we consider and the approach that we adopt. Consider the (relational) database of Figure 1.1(a) which contains two tuples: the tuple ab (over scheme AB) and the tuple bc (over scheme BC). Moreover, consider the following query on that database:

Q : List all tuples over scheme ABC.

Relational theory will always produce (by taking a join) the same answer: $\alpha(Q) = \{roman\ a, b, c\}$, *no matter whether functional dependencies are given or not!* In other words, query answering in the relational model is *insensitive* to semantic information.

A	B
a	b

B	C
b	c

(a)

I(a) = {1,2,3}
I(b) = {2,3,4} (b)
I(c) = {4,5}

I(ab) = I(a) ∩ I(b) = {1,2,3} ∩ {2,3,4} = {2,3} (c)
I(bc) = I(b) ∩ I(c) = {2,3,4} ∩ {4,5} = {4}

FIGURE 1.1 Interpreting a relational database

Let us now introduce our approach. First, we consider an *interpretation* I, that is, a function assigning a set of integers (possibly empty) to every symbol in the domains of A,B, and C. However, we require that the function I assign a nonempty set of integers to every such symbol appearing in the database. An example is shown in Figure 1.1(b), where it is assumed that $I(x)=\phi$ for all x not in {a,b,c}. Next, we extend the interpretation I to tuples as follows: for any tuple t, if $t = x_1x_2 \cdots x_n$ then $I(t) = I(x_1) \cap I(x_2) \cap \cdots \cap I(x_n)$. Now, an interpretation is a *model* of our database if it assigns a non empty set to every tuple appearing in the database. The interpretation of Figure 1.1(b) is indeed a model for the database; this is verified in Figure 1.1(c). Finally, suppose that, in answering a query, we include in the answer only those tuples that have a nonempty interpretation in *every* model of our database. Then the answer to the query Q formulated earlier is empty! Indeed, if we consider the model of Figure 1.1(b), then we find:

$$I(abc) = I(a) \cap I(b) \cap I(c) = \{1,2,3\} \cap \{2,3,4\} \cap \{4,5\} = \phi$$

and, clearly, for any other tuple t over ABC such that $t \neq abc$, we can find a model I' such that $I'(t) = \phi$. 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: $I(b) \subseteq I(c)$, then we can *infer* that $I(abc)$ is nonempty, in *every* admissible model. Indeed, in every admissible model we have: $I(ab) \neq \phi$, by definition; on the other hand, $I(b) \subseteq I(c)$ implies that: $I(b) \cap I(c) = I(b)$. Combining these two facts we obtain:

$$I(abc) = I(a) \cap I(b) \cap I(c) = I(a) \cap I(b) = I(ab) \neq \phi$$

It follows that:

if $I(b) \subseteq I(c)$ then $\alpha(Q) = \{abc\}$.

(We shall see later that the constraint: $I(b) \subseteq I(c)$ corresponds to the functional dependency $B \rightarrow C$.) Thus the approach suggested here improves over the relational approach, in the sense that query processing becomes sensitive to semantic constraints.

A similar argument holds for update processing. Indeed, consider updating the database of Figure 1.1, say by inserting the tuple abc. If there is no constraint, then the tuple abc cannot be *inferred* from ab and bc, in the sense that we cannot prove that I(abc) $\neq \phi$ in every model. Therefore, in this case, the tuple abc must be stored in the database (we shall see how later on). On the other hand, if a constraint is given, say I(b) $\subseteq$ I(c), then the tuple abc can be *inferred* from ab and bc (in the way that we saw earlier), and we may choose not to store it in the database.

In Section 2 of the paper we give a brief description of our model and give formal definitions of the concepts that we have just outlined here, namely *interpretation, model, and truth* and give algorithms for the following inference problem: Given a database D, (a) determine the set of all tuples implied by D and (b) given tuples s and t, determine whether s implies t. In Section 4, we discuss equivalence of databases. Two databases are called equivalent if they imply the same set of tuples. We present an algorithm for testing equivalence. In Section 5, we discuss query answering. In Section 5.1, we present our semantic approach to query answering, and in Section 5.2, we compare it to the syntactic approach of the relational model. In Section 6, we discuss update processing, namely insertion and deletion of tuples in a database. In contrast with the relational model, we study updating of equivalence classes rather than of particular databases, and we argue that our approach can lead to a deeper understanding of update semantics. Finally, in Section 7, we discuss some limitations of our approach and indicate directions for future research. Throughout the paper, we assume familiarity with the basic relational terminology (as in [12]).

2. THE MODEL

The data model that we adopt for our investigations considers the relation schemes, the constraints, and the database, as strings of uninterpreted symbols. Interpretations for these symbols are provided using subsets of an underlying population of objects. Accordingly, our data model comprises three components: a syntactic component, which is essentially the relational model; a semantic domain,

which is the power set of an underlying population of objects; and an interpretation component, which provides the link between the syntactic component and the semantic domain.

Syntax

We begin with a finite, nonempty set $U = \{A_1, A_2, \ldots, A_n\}$. The set U is called the *universe* and the A_i's are called *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 $U \cap Dom(A_i) = \phi$ for all i, and that $dom(A_i) \cap dom(A_j) = \phi$ for $i \neq j$ (we shall discuss the consequences of this assumption in Section 7). A *relation scheme* over U is a nonempty subset of U; 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 $t(A_j)$ is in $dom(A_j)$, for all A_j 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 attributes in R. If t is a tuple over $R = \{A_1, A_2, \ldots, A_k\}$, and if $t(A_j)=a_j$ then we denote the tuple t as $a_1 a_2 \cdots a_k$. A *relation* over R is a set of tuples over R. Thus a relation over R is a subset of dom(R). 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}, and let
$dom(A) = \{a_1, a_2, \ldots\}$, $dom(B) = \{b_1, b_2, \cdots\}$, $dom(C) = \{c_1, c_2, \ldots\}$
Define a function δ on relation schemes over U as follows :

$\delta(AB) = \{a_1b_1, a_1b_2, a_2b_1\}$
$\delta(BC) = \{b_2c_1, b_3c_1\}$
$\delta(R) = \phi$, 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 the 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. ▫

A remark is in order here. Given a database $D=(\delta, \Sigma)$, call *database scheme* the set of all relation schemes that are assigned non-empty relations under δ. That is,

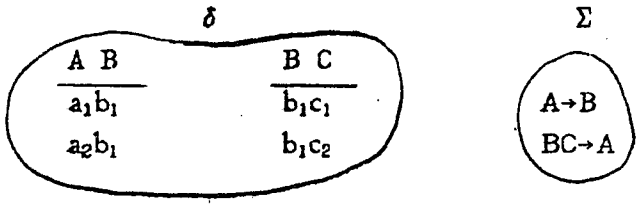
the relation scheme of D is the set $\{R \subseteq U \,/\, R \neq \phi \,,\, \delta\,(R) \neq \phi\}$. In the relational literature, the database scheme is considered *fixed* and constitutes part of the definition of a database. The reasons for fixing the database scheme are mostly related to implementation issues. However, the relational literature abounds in artificial problems that are directly (or solely) due to the assumption of a fixed database scheme. Null values of the type "value does not exist", or view updating are just two examples of such artificial problems. The assumption of a fixed database scheme is all the less justified in a universal relation interface, where the user is not at all aware of the grouping of attributes into relation schemes. Accordingly, we do *not* make the assumption of a fixed database scheme in our approach.

$U = \{A,B,C\}$

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

(a)
$I(a_1) = 13$
$I(a_2) = 24$
$I(b_1) = 1234$
$I(b_2) = 56$
$I(c_1) = 3$
$I(c_2) = 1$
$I(x) = \phi$, for every x different than a_1 , a_2 , b_1 , b_2 , c_1 , c_2.

(b)



(c)
$I(a_1b_1) = I(a_1) \cap I(b_1) = 13$
$I(a_2b_1) = I(a_2) \cap I(b_1) = 24$
$I(b_1c_1) = I(b_1) \cap I(c_1) = 3$
$I(b_1c_2) = I(b_1) \cap I(c_2) = 1$
$I(a_1b_1c_1) = I(a_1) \cap I(b_1) \cap I(c_1) = 3$
$I(a_1b_1c_2) = I(a_1) \cap I(b_1) \cap I(c_2) = 1$

FIGURE 2.1 An interpretation I of U={A,B,C}, and a database $D = (\delta , \Sigma)$ over U, for which I is a model.

Given a database (δ , Σ), such as the one of Figure 2.1(b), the relation schemes, the relations and the dependencies are regarded as uninterpreted symbols. Their *meaning* (or semantics) is specified by associating to them objects from a set that we are about to define.

The semantic domain P_*

We assume that the "real world" consists of a countably infinite set of objects, and we identify these objects to the positive integers. Let ω be the set of all

positive integers, and let $P_\omega = \{\tau \;/\; \tau \subseteq \omega\}$ be the set of all subsets of ω. The set P_ω is the semantic domain in which relation schemes, relations, and dependencies receive their interpretations. In order to avoid proliferation of brackets, we adopt the following notational convention: a set of integer is denoted by juxtaposition of its elements in increasing order. For example, the set {3,5,8} is denoted as 358. (As we shall never use examples with more than nine integers, this creates no confusion).

Interpretations

Throughout our discussions, we consider fixed the universe of attributes $U=\{A_1,A_2,\ldots,A_n\}$, 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,

$$SYMBOLS = \bigcup \{\, dom(A) \;/\; A \in U \,\}$$

$$TUPLES = \bigcup \{\, dom(R) \;/\; R \subseteq U,\ R \neq \phi \,\}$$

Clearly, SYMBOLS is a subset of TUPLES.

Definition 2.1 An *interpretation* of U is a function I from SYMBOLS into P_ω such that,

$$\forall A \in U \;\; \forall a,a' \in dom(A),\ (\, I(a) \cap I(a') = \phi \;\vee\; a = a' \,). \qquad \square$$

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 7).

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

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

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 $I(ab) = I(a) \cap I(b)$. Notice that the basic property of an interpretation is

satisfied by the extension:

$$\forall R \subseteq U \ \forall t,t' \in dom(R) \ (I(t) \cap I(t') = \phi \ \vee \ t = t')$$

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

Definition 2.2 Let I be an interpretation of U. A tuple t in TUPLES is called *true* in I if $I(t) \neq \phi$; otherwise, t is called *false* in I. ▫

Note that if a tuple, say abc, is true in I, then all its "subtuples", namely a,b,c,ab,ac and bc, are also true in I. However, it is important to note that if two tuples, say ab and bc are true in an interpretation I, their "join" abc is not necessarily true in I ! For example, in Figure 2.1(c) we see that tuples a_2b_1 and b_1c_1 are true, while tuple $a_2b_1c_1$ is false.

Intuitively speaking, any given database (δ , Σ) over U is assumed to represent true information about the real world. Therefore, only interpretations of U that *do* conform to this assumption are to be considered. Hence the following definition:

Definition 2.3 Let $D = (\delta , \Sigma)$ be a database over U. An interpretation I of U is called a *model* of D if

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

(2) $\forall X \rightarrow Y \in \Sigma \ \forall x \in dom(X) \ \forall y \in dom(Y)$
$I(x) \cap I(y) \neq \phi \implies I(x) \subseteq I(y)$ ▫

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 $X \rightarrow Y$ in Σ (in the sense that the set of pairs $\{ (x,y) / x \in dom(X), y \in dom(Y), I(x) \cap I(y) \neq \phi \}$ is a function).

In Figure 2.1 we see an interpretation I and a database (δ , Σ). Using the computations of Figure 2.1(c) we 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 nonempty interpretations under I.

(2) For $A \rightarrow B$: the only true tuples over AB 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 $A \rightarrow B$ is satisfied.

For BC→A: the only true tuples over ABC are $a_1b_1c_1$, $a_1b_1c_2$, 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.

Clearly, such verifications are possible only if the set of symbols that are assigned nonempty interpretations under I is finite. We call such an interpretation a *finite interpretation*. As we shall see in the following sections, finite interpretations are sufficient to capture query and update semantics in relational databases. The reason for this is that the symbols really of interest are those appearing in the database, and their number is finite.

Note that the data model adopted in this paper allows also the specification of dependencies at a "lower level" than functional dependencies, that is, at tuple level. Indeed, if x_0 is a given value in dom(X_0) and y_0 a given value in dom(Y_0) then we may require that

$I(x_0) \cap I(y_0) \neq \phi \implies I(x_0) \subseteq I(y_0)$

only for x_0 and y_0. This kind of conditions are the counterpart of those used in conditional tables [7]. In fact *any* expression on interpretations formed by union, intersection, difference and set-inclusion can be considered as a "constraint". We shall not consider this kind of constraints here. The interested reader is referred to [11].

3. SEMANTIC IMPLICATION

Given a universe U, databases over U can be seen as restrictions, on the interpretations of U, in the following sense: with repect to a given database, some of the interpretations are models and some are not. In particular, some databases may not have models at all. A database is called *semantically inconsistent* if it possesses no model, and otherwise it is called *semantically consistent*. In our discussions we shall always assume that databases are semantically 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 apeear 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 3.1 Let $D = (\delta, \Sigma)$ 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. If T is a set of tuples then D *implies* T, denoted $D \models T$, if $D \models t$ for all t in T. ▫

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. 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 $I'(a_1) = 24$, $I'(a_2) = 13$, 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 define

$$T(D) = \bigcap \{ T(m) \ / \ m \text{ is a model of } D \}$$

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

$$\forall t \in \text{TUPLES} \ , \ D \models t \iff t \in T(D)$$

We define a relation between tuples s and t of T(D) as follows :

$$s \models_D t \text{ iff } m(s) \subseteq m(t), \quad \text{for every model m of } D$$

(The subscript D is omitted when no confudsion is possible). For example, in Figure 3.1, we have : $ab \models a$, which is obvious. What is less obvious is that $ab \models abc$. Indeed, in every model m of D, we have : $m(bc) \neq \phi$, by definition of a model, and $m(b) \subseteq m(c)$, because of B→C. It follows that : $m(ab) \subseteq m(abc)$. By the way, this example shows that the relation $\models$ is *not* antisymmetric (indeed, $ab \models abc$ and $abc \models ab$ but $ab \neq abc$). Nevertheless it is a reflexive and transitive relation. Also, it is not difficult to see that, for any database D and tuples s and t, if $D \models s$ and $s \models t$ then $D \models t$.

If $s \models t$ and $t \models s$, we write $s \models\!\!\models t$ and call s and t *semantically equivalent*. For example, in Figure 3.1, we have: $ab \models\!\!\models abc$. Clearly, the relation $\models\!\!\models$ is an equivalence relation on the set T(D), and we can define a relation between equivalence classes C_1 and C_2 as follows:

$$C_1 \leqslant C_2 \iff \exists s \in C_1 , \exists t \in C_2 , s \models t$$

Clearly, the relation $\leqslant$ is a partial order on equivalence classes and it is not

difficult to see that every minimal equivalence class contains a database tuple.

In the remaining of this section, we consider the following inference problems. Given a database $D=(\delta, \Sigma)$,

(1) determine the set T(D), that is, determine the set of all tuples in TUPLES implied by D, and

(2) determine wether $s \models t$, for tuples s and t in TUPLES.

Our approach for solving these problems can be summerized as follows:

(1') Find a model m_q of D such that $T(D)=T(m_q)$; it will follow that: t is in T(D) iff t is true in m_q, (for any t in TUPLES).

(2') Find a model m_u of D such that $T(D)=T(m_u)$, and $s \models t$ iff $m_u(s) \subseteq m_u(t)$.

The model m_q will allow us to test the inference $D \models t$. In Section 5, we shall use m_q in order to compute answers to queries as well. Accordingly, we shall call m_q a *query model*. Similarly, the model m_u will allow us to test the inference $s \models t$. In Section 6, we shall use m_u in order to compute results of updates as well. Accordingly, we shall call m_u an *update model*. Before giving algorithms for constructing query and update models, we need some formal definitions.

Definition3.2 Let D be a database. A model m_q of D is called a *query model* of D if $T(m_q)=T(D)$ □

In order to define update models, we need the concept of inseparability.

Definition 3.3 Let D be a database, and m a model of D. Two integers i and j are called *inseparable* in m if

(1) $\exists$ a $\in$ SYMBOLS , $\{i,j\} \subseteq m(a)$

(2) $\forall$ b $\in$ SYMBOLS , $i \in m(b)$ iff $j \in m(b)$

Definition 3.4 Let D be a database. A model m_u of D is called an *update model* of D if

(1) $T(m_u) = T(D)$,

(2) there is no pair of integers inseparable in m_u,

(3) $\forall t_0, t_1, \ldots, t_n \in$ TUPLES,

$m_u(t_0) \subseteq m_u(t_1) \cup \cdots \cup m_u(t_n) \Rightarrow \exists i \in \{1,2,\ldots,n\}, t_0 \models t_i$ □

The first condition says that m_u is a query model. The second condition says that m_u is not redundant. Indeed, according to Definition 3.3, inseparable integers either appear together in the interpretation of a symbol or they do not appear at all. Thus the presence of inseparable integers would imply redundancy. Finally, the following lemma will help explain the third condition of definition 3.4.

Lemma 3.1 Let D be a database and m a model of D. Then the following properties are equivalent:

(1) $\forall t_0, t_1, \ldots, t_n \in \text{TUPLES}$,
$m_u(t_0) \subseteq m_u(t_1) \cup \cdots \cup m_u(t_n) \implies \exists i \in \{1,2,\ldots,n\}, t_0 \models t_i$

(2) $\forall t_0 \in T(D), \exists k_{t_0} \in m_u(t_0), \forall t \in T(D), k_{t_0} \in m_u(t) \implies t_0 \models t$ □

Proof

- The implication (2) $\implies$ (1) is easy.
- Let us show that $\neg$ (2) $\implies \neg$ (1).

Suppose that (2) is not verified. Then, there is t_0 in T(D), such that

$\forall k \in m_u(t_0), \exists t_k \in T(D), k \in m_u(t_k) \wedge t_0 \models_D t_k$.

Then, we have the inclusion $m_u(t_0) \subseteq \bigcup \{m_u(t_k) / k \in m_u(t_0)\}$ and also $\forall k \in m_u(t_0), t_0 \models_D t_k$. We can conclude from these properties that (1) is not satisfied. □

What property (2) of this lemma says is that there is a one-to-one correspondence between the tuples of T(D) and the integers appearing in m. Speaking roughly, the model m "counts" the tuples of T(D). Thus, in view of Lemma 3.1, Definition 3.4 can be stated (roughly) as follows: an update model is a query model which contain no pair of inseparable integers, and counts the tuples of T(D). Now, as the integers appearing in an update model are in one-to-one correspondence with the integers appearing in the update model for D (up to renaming of the integers). This is stated formally in the following theorem. In order to simplify matters, we call a model m *irreducible* if there is no pair of integers inseparable in m.

Theorem 3.1 Let D be a database. If there is an update model of D, then it is unique (up to a renaiming of the integers). □

Proof

Let m be the model defined by $m(a) = \{ k_t , t \models_D a \}$. It is clear that $m(a) \subseteq m_u(a)$ for all a because $k_t \in m_u(t)$ by definition of k_t, and so $k_t \in m_u(a)$ because $t \models_D a$. Let i be an element of $m_u(a)$. Let K be the set of all tuples such that $i \in m_u(t)$. Let t_0 be a tuple in K such that $m_u(t_0)$ is minimal in K (for the $\subseteq$ order). Then, we can see that i and k_{t_0} are inseparable. Indeed, $\{i, k_{t_0}\} \subseteq m_u(t_0)$ and there is no tuple t such that i is in $m_u(t)$ and k_{t_0} is not, because of the minimality of t_0. We conclude that $k_{t_0} = i$ because m_u is irreducible. From lemma 3.1, we have $t_0 \models_D a$, so i is in m(a). So we have the inclusion

$m_u(a) \subseteq m(a)$ and then $m_u = m$. We can conclude from this equality that the update model m_u is unique up to a renaming of the integers k_t chosen. □

We have defined so far query and update models, and we have seen an interesting property of update models, namely that they are unique. We now proceed to show that such models indeed exist, by giving algorithms for their construction. In order to simplify the presentation, we assume that

(1) All dependencies in Σ are of the form $X \rightarrow A$, where X is any relation scheme and A is a single attribute.
(2) All non-trivial dependencies of the form $X \rightarrow A$ which are implied by Σ (in the sense of the relational model) are in Σ.

Let us note that any set Σ' of functional dependencies can be transformed into an equivalent set Σ satisfying conditions (1) and (2) above (see in [12]). We also assume that Σ contains no trivial dependencies.

The following algorithm computes a query model of a given database D.

Algorithm 3.1 Query Model
Input : A database $D = (\delta , \Sigma)$
Output: A model m_q of D such that $T(m_q) = T(D)$

(0) Assign a distinct positive integer i_t to every database tuple t.
Set $m_1(a) = \phi$ for all a in SYMBOLS.

(1) For every database tuple $t=a_1\ a_2 \cdots a_k$, add the integer i_t to each of the sets $m_1(a_1)$, $m_1(a_2)$, . . . , $m_1(a_k)$.

(2) For $j>=1$, compute m_{j+1} from m_j as follows until $m_{j+1} = m_j$:
if there is $X \rightarrow A \in \Sigma$, $x \in dom(X)$, $a \in dom(A)$ *such that*
$m_j(x) \cap m_j(a) \neq \phi$ *and* $m_j(x) \not\subseteq m_j(a)$
then set $m_{j+1}(a) = m_j(x) \cup m_j(a)$ *and*
for all $a' \neq a$ in SYMBOLS set $m_{j+1}(a') = m_j(a')$
else set $m_{j+1} = m_j$

(3) Set $m_q = m_{j_0}$, where m_{j_0} is the last function computed at Step 2. □

Example 3.1 Let us apply this algorithm to the database $D_1 = (\delta , \Sigma_1)$ shown in

Figure 3.1.

Step 0 : Each database tuple is assigned an integer, for example, $a_1b_1c_1 \to 1$, $a_2b_2c_1 \to 2$, $a_1b_1d_1 \to 3$, $a_3b_2d_1 \to 4$.

Step 1 : The symbols appearing in the database are: a_1, a_2, a_3, b_1, b_2, c_1, d_1. Model m_1 assigns nonempty sets of integers to these symbols as follows: $m_1(a_1) = 13$, because a_1 appears in the tuples numbered 1 and 3, $m_1(a_2) = 2$, because a_2 appears in the tuple numbered 2, and so on. In this way we find

m_1: $a_1 \to 13$, $a_2 \to 2$, $a_3 \to 4$, $b_1 \to 13$, $b_2 \to 24$, $c_1 \to 12$, $d_1 \to 34$

Step2 : The dependency A→B does not modify m_1 as the condition for modification is false for all tuples over AB. The dependency B→C *does* modify m_1. Indeed, as $m_1(b_1) \cap m_1(c_1) = 1$ and $m_1(b_1) \not\subseteq m_1(c_1)$, we define m_2 as follows:

$m_2(c_1) = m_1(b_1) \cup m_1(c_1) = 123$, $m_2(x) = m_1(x)$ for all $x \neq c_1$.

Similarly, as $m_2(b_2) \cap m_2(c_1) = 2$ and $m_2(b_2) \not\subseteq m_2(c_1)$ we define m_3 as follows:

$m_3(c_1) = m_2(b_2) \cup m_2(c_1) = 1234$, $m_3(x) = m_2(x)$ for all $x \neq c_1$.

No further modification is possible based on dependencies B→C or A→B. Finally, the dependency A→C does not modify m_3, as the condition for modification is false, for all tuples over AC.

Step 3 : The model $m_{q_1} = m_3$ is a query model of D_1 (and it is shown in Figure 3.1). □

δ

A	B	C
a_1	b_1	c_1
a_2	b_2	c_1

A	B	D
a_1	b_1	d_1
a_3	b_2	d_1

$\Sigma_1 = \{ A \to B , B \to C , A \to C \}$
$\Sigma_2 = \Sigma_1 \cup \{ A \to D , B \to D \}$
$\Sigma_3 = \Sigma_1 \cup \{ BD \to A , BD \to C \}$

m_{q_1}	m_{q_2}	m_{q_3}
$a_1 \to 13$	$a_1 \to 13$	$a_1 \to 13$
$a_2 \to 2$	$a_2 \to 2$	$a_2 \to 2$
$a_3 \to 4$	$a_3 \to 4$	$a_3 \to 4$
$b_1 \to 13$	$b_1 \to 13$	$b_1 \to 13$
$b_2 \to 24$	$b_2 \to 24$	$b_2 \to 24$
$c_1 \to 1234$	$c_1 \to 1234$	$c_1 \to 1234$
$d_1 \to 34$	$d_1 \to 1234$	$d_1 \to 34$

FIGURE 3.1 Three databases $D_1 = (\delta , \Sigma_1)$, $D_2 = (\delta , \Sigma_2)$, $D_3 = (\delta , \Sigma_3)$, and their query models m_{q_1} , m_{q_2} , m_{q_3} produced by algorithm 3.1.

Theorem 3.2 Algorithm 3.1 terminates and if m_q is an interpretation of U then m_q is a query model of D, else D is inconsistent. □

Proof

- Algorithm 3.1 clearly terminates.
- Let us suppose that $m_1 , \cdots , m_{j_0}$ are interpretations. We will show by induction that every m_j satisfies the following property : $T(m_j) \subseteq T(D)$.
  - The property is easily verified for the interpretation m_1.
  - Suppose that $T(m_j) \subseteq T(D)$ and let f be a tuple such that f is in $T(m_{j+1})$ but not in $T(m_j)$. Let $X \to A$, x and a be the elements used to build m_{j+1} from m_j at Step 2. Then m_j and m_{j+1} are equal except on a. So a is necessarily a subtuple of f and we can write f = af'. Then, we have :

$$\begin{aligned} m_{j+1}(f) &= m_{j+1}(a) \cap m_{j+1}(f') \\ &= (m_j(a) \cup m_j(x)) \cap m_j(f') \\ &= m_j(af') \cup (m_j(x) \cap m_j(f')) \\ &= m_j(x) \cap m_j(f') \neq \phi . \end{aligned}$$

  Let m be a model of D; we have $m(x) \cap m(f') \neq \phi$ because $T(m_j) \subseteq T(D)$, and also $m(x) \subseteq m(a)$ because $X \to A$ is in Σ. Combining these two facts, we obtain that $m(a) \cap m(f') = m(f) \neq \phi$, that is, f is in $T(D)$.

In conclusion, if m_q is an interpretation then $T(m_q) \subseteq T(D)$. As m_q is clearly a

model of D, we also have $T(D) \subseteq T(m_q)$ and thus $T(m_q) = T(D)$. It follows that m_q is a query model.

- If m_q is not an interpretation, then suppose that D is consistent and let j be the highest integer such that $m_1, \ldots, m_j$ are interpretations. (m_1 is clearly always an interpretation). Let $X \rightarrow A$, x and a be the elements used to build m_{j+1} from m_j. There is a symbol a' in dom(A) such that $m_{j+1}(a) \cap m_{j+1}(a') \neq \phi$. So we have $m_j(a') \cap m_j(x) \neq \phi$ (because m_j is an interpretation and because of the definition of m_{j+1}) and of course $m_j(a) \cap m_j(x) \neq \phi$. So a' x and a x are in $T(m_j)$ and so in $T(D)$ because $m_1, \cdots, m_j$ are interpretations. This is impossible because $X \rightarrow A$ is in Σ. So D is inconsistent. ▫

The following algorithm computes the update model of a given database D.

Algorithm 3.2 Update Model

Input: A database $D = (\delta, \Sigma)$

Output: A model m_u of D such that m_u is an update model of D.

(0) -Assign a distinct positive integer i_t to every sub-tuple t of a tuple of the database.
-Set $m_1(a)$ to the empty set for all a in SYMBOLS.

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

(2) -For $j \geq 1$, compute m_{j+1} from m_j as follows, until $m_{j+1} = m_j$:
If there is $X \rightarrow A \in \Sigma$, $x \in$ dom(X) and $a \in$ dom(A) such that
$m_j(x) \cap m_j(a) \neq \phi$ and $m_j(x) \not\subseteq m_j(a)$,
then
- compute m_j' as follows:
$m_j'(a) = m_j(a) \cup m_j(x)$,
$m_j'(b) = m_j(b)$, for every b in SYMBOLS such that $b \neq a$, and
- m_{j+1} is the interpretation obtained from m_j' as follows:
For every tuple $t=a_1 \cdots a_k$ such that $t \, m_j'(t) \neq \phi$ and $m_j(t) \neq \phi$, add a new integer i_t to $m_j'(a_p)$ for all p in {1...k}.
else let m_{j+1} be m_j.

(3) -Let m_{i_0} be the last function computed at Step 2. Let m_u be the irreducible model obtained by removing from m_{i_0} an element of

each pair of inseparable integers. □

The following theorem shows that Algorithm 3.2 computes an update model of the given database.

Theorem 3.3 Algorithm 3.2 always terminates and if m_u is an interpretation of U then m_u is an update model of D, else D is inconsistent. □

Proof

Let $D'=(\delta', \Sigma)$ be the database obtained from D by adding to δ all the subtuples of the tuples in δ. It is clear that D' is consistent iff D is consistent and that $T(D') = T(D)$. So, comparing Algorithms 3.1 and 3.2, we can conclude from Theorem 3.2 that m_u is an interpretation of U iff D is consistent and ,if it is, that m_u is a query model of D. It is also clear that m_u is irreducible. So, in order to show that m_u is an update model, we just have to show the following property :

(H) $\forall t_0 \in T(D)\ ,\ \exists k \in m_u(t_0)\ ,\ \forall t \in T(D)\ ,\ k \in m_u(t) \implies t_0 \models_D t$.

Let us consider the following induction assumption :

(H_i) $\forall t_0 \in T(m_i)\ ,\ \exists k \in m_u(t_0)\ ,\ \forall t \in T(D)\ ,\ k \in m_i(t) \implies t_0 \models_D t$.

- (H_1) is verified. Indeed, for every t_0 in $T(m_1)$, t_0 is a sub-tuple of a tuple of the database and then, we can take $k = i_{t_0}$ where i_{t_0} is the integer associated with t_0 in the step (0) of the algorithm.

- Let us suppose that (H_i) is verified. Let $X \to A$, x and a be the elements used to compute m_{i+1} from m_i. Let t_0 be a tuple in $T(m_{i+1})$.

1) Suppose that t_0 is also in $T(m_i)$, then let k be the integer associated with t_0 in property (H_i). Let t be in T(D) such that $k \in m_{i+1}(t)$.

- if $k \in m_i(t)$, we clearly have $t_0 \models_D t$.

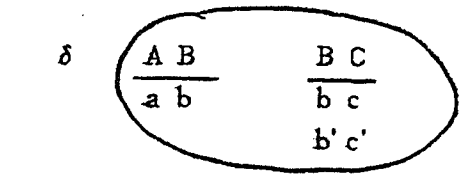
- else k is in $m_i'(t)$ and so, a is a sub-tuple of t. We denote $t=at'$. Now we have $m_i'(t)=m_i(t) \cup m_i(x) \cap m_i(t')$ by definition of m_i'. So k is in $m_i(x)$ and in $m_i(t')$ and we conclude from (H_i) that $t_0 \models_D x$ and $t_0 \models_D t'$. We clearly have $x \models_D a$, and we finally obtain $t_0 \models_D t$.

2) Suppose that t_0 is in $T(m_i')$ but not in $T(m_i)$, we can show that there is a tuple t_1 in $T(m_i)$ such that $t_0 \models_D t_1$, and we are in case 1) with t_1 instead of t_0.

3) If t_0 is not even in $T(m_i')$, then t_0 has been associated with an integer i_{t_0} in step (2) of the algorithm, and, if $i_{t_0} \in m_{i+1}(t)$, thent is necessarily a sub-tuple of t_0, so we have $t_0 \models_D t$.

In conclusion, the condition (H_{i+1}) is verified, and then, the condition (H_i) is verified for every i. As $m_u = m_{i_0}$ for some i_0, then the condition (H) is also

verified. So m_u is an update model of D. □



m_q	a → 1	m_u	a → 139
	b → 13		b → 12
	b' → 2		b' → 4
	c → 13		c → 1289
	c' → 2		c' → 46

Figure 3.2 A database D=(δ , Σ) and associated query and update models.

Example 3.2 Let us apply the Algorithm 3.2 to the database D of Figure 3.2.

Step 0: Each subtuple of a database tuple is associated with an integer, for example:

ab :	1	b :	2
a :	3	b'c':	4
b' :	5	c' :	6
bc :	7	c :	8

Step 1: We have $m_1(a)=13$ because a is in the tuples ab and a associated with the integers 1 and 3, respectively. In the same way we find:

m_1: a → 13, b → 127, b' → 45, c → 78, c' → 46.

Step 2: We compute $m_2, \cdots$ as follows:

- $B \to C \in \Sigma$, and $m_1(b) \cap m_1(c) \neq \phi$, and $m_1(b) \not\supseteq m_1(c)$, so

m_1': a → 13, b → 127, b' → 45, c → 1278, c' → 46.

We can see now that $m_1'(t) \neq \phi$ and $m_1(t) = \phi$ only for t=ac, so

m_2: a → 139, b → 127, b' → 45, c → 12789, c' → 46.

- $B \to C \in \Sigma$, and $m_1(b') \cap m_1(c') \neq \phi$, and $m_1(b') \not\supseteq m_1(c')$, so

m_2': a → 139, b → 127, b' → 45, c → 12789, c' → 456.

We can remark that $m_3 = m_2'$.

No further modification is possible based on dependency B→C.

Step 3 The pairs of inseparable integers are {2,7} and {4,5}. So the model m_u computed by Algorithm 3.2 is the model m_u shown in Figure 3.2.

In the following sections, we use the concepts of query model and update model, to define equivalence between databases, query answering and update processing.

4. EQUIVALENCE

In this section, we study equivalence between databases. Intuitively, two databases are equivalent if they imply exactly the same set of tuples. We state this formally in the following definition.

Definition 4.1 Let D and D' be two databases. We say that D is *smaller* than D', denoted by $D \prec D'$, if $T(D) \subseteq T(D')$. We say that D is *equivalent* to D', denoted by $D \equiv D'$, if $T(D) = T(D')$ ▫

Let BASES(Σ) be the set of all databases on a given universe U with a given set Σ of functional dependencies. Clearly, the relation ≡ is an equivalence relation on the set BASES(Σ). We denote by $\overline{D}$ the equivalence class of database D, and we denote by BASES(Σ)/≡ the set of all equivalence classes of the set BASES(Σ). It follows from Definition 4.1 that there is a one-to-one correspondence between the set BASES(Σ)/≡ and the set {T(D) / D ∈ BASES(Σ)}. By the way, we can define a relation between equivalence classes C and C' as follows:

$C \leqslant C'$ iff $\exists D \in C,\ \exists D' \in C',\ D \prec D'$

Clearly, the relation ⩽ is a partial order on the set BASES(Σ)/≡, corresponding to set inclusion on the set {T(D) / D ∈ BASES(Σ)}.

It is important to note that the databases in an equivalence class, say $\overline{D}$, are just different representations of the same information, namely of the set T(D). Now, depending on the situation, one can be interested only in the informationcontained in an equivalence class, ignoring representational details, or one can be interested in a particular representation of the equivalent class for varoius reasons. This will become clear in the following section, where we discuss query answering in a universal relation interface. In such an environment and from the user viewpoint, it is immaterial which representation is used, as long as the answers are the same. However, from the system viewpoint, some representations are more convenient than others, and we are faced with a choice between equivalent representations of the same information. Therefore, the problem of

determining equivalence between databases is important.

In the previous section, we have seen that every consistent database posseses an update model which is unique, up to renaming of the integers. Let us recall that a renaming of the positive integers is a bijective function from the set of (positive) integers ω into itself. Now, two interpretations I and I' are equivalent, denoted by $I \equiv I'$, if there is a renaming ρ such that $\rho(I(x)) = I'(x)$, for all x in SYMBOLS.

Theorem 4.1 Let D and D' be databases, with update models m_u and m_u', repectively. Then

$D \equiv D'$ iff $m_u \equiv m_u'$ ▫

Proof Easy.

Thus, two databases are equivalent if and only if their update models are equivalent. The following algorithm determines equivalence of update models. We denote by $|m|$ the set of integers appearing in model m, that is, $|m| = \bigcup \{m(a) / a \in SYMBOLS\}$. We denote by $P_{|m|}$ the set of all subset of $|m|$, and we denote by $m^{-1}(\alpha)$ the set $\{a \in SYMBOLS , \alpha \in m(a)\}$.

Algorithm 4.1 Equivalence of Update Models.

Input : two update models m_u and m_u'.

Output: a boolean value.

(0) We build a function Ψ from $|m|$ into $P_{|m|}$ as follows:
- First let $\Psi(\alpha)$ be ω, for all α in $|m|$.
- Then for every a in SYMBOLS, such that $\alpha \in m_u(a)$,
  replace $\Psi(\alpha)$ by $\Psi(\alpha) \cap m_u'(a)$.

(1) Let T_0 and S_0 be the empty set.

(2) We compute T_{i+1} and S_{i+1} from T_i and S_i as follows until $T_{i+1} = T_i$ and $S_{i+1} = S_i$:
if there is α such that $\Psi(\alpha) - T_i$ is a singleton $\{\beta\}$, and $m_u^{-1}(\alpha) = m_u'^{-1}(\beta)$,
then $T_{i+1} = T_i \cup \{\beta\}$, and $S_{i+1} = S_i \cup \{\alpha\}$
else $T_{i+1} = T_i$ and $S_{i+1} = S_i$.

(3) Let T and S be the last sets T_{i_0} and S_{i_0} computed at Step 2.
if $T = |m_u'|$ and $S = |m_u|$ *then* the output is YES
else the output is NO ▫

The following theorem proves the correctness of Algorithm 4.1.

Theorem 4.2 Algorithm 4.1 terminates and two update models m_u and m_u' are equivalent if and only if the output of Algorithm 4.1 is YES

Proof

Let us denote by m_i and m_i' the models defined from m_u by

(1) $m_i(a) = m_u(a) - S_i$, for all a in SYMBOLS.

(2) $m_i'(a) = m_u'(a) - T_i$, for all a in SYMBOLS.

In order to show the theorem, we prove the following property for all i:

(H_i) $m_i \equiv m_i'$ iff $m_{i+1} \equiv m_{i+1}'$

1) Let us suppose that $m_i \equiv m_i'$, then if $S_{i+1} = S_i$ and $T_{i+1} = T_i$ then the result is obvious but else there is α such that $\Psi(\alpha) - T_i = \{\beta\}$ and $m_u^{-1}(\alpha) = m_u'^{-1}(\beta)$. So, the bijection ρ between $|m_i|$ and $|m_i'|$ verifies the equality $\rho(\alpha) = \beta$. Indeed, as m_u (and so m_i) is irreducible, then there is a tuple t such that $m_i(t) = \alpha$. By definition of Ψ, we also have $m_i'(t) = \beta$. So ρ induces a bijection from $|m_{i+1}|$ into $|m_{i+1}'|$ and so $m_{i+1} \equiv m_{i+1}'$.

2) Let us suppose, now that $m_{i+1} \equiv m_{i+1}'$. Suppose also that $S_{i+1} \neq S_i$ and $T_{i+1} \neq T_i$. Let α and β be the integers such that $T_{i+1} = T_i \cup \{\beta\}$ and $S_{i+1} = S_i \cup \{\alpha\}$. There is a bijection ρ_{i+1} from $|m_{i+1}|$ into $|m_{i+1}'|$. Let us define ρ_i as follows:

$\rho_i(x) = \rho_{i+1}(x)$, if $x \neq \alpha$, and

$\rho_i(\alpha) = \beta$.

First of all, ρ_i is obviously a bijection. It is not difficult to see that β is the only element of the set $\bigcap \{m_i'(b), \alpha \in m_i(b), b \mid \in \text{SYMBOLS}\}$, and we can conclude easily that $\rho_i(m_i(a)) \subseteq m_i'(a)$. Let k be an element of $m_i'(a)$, if $k \neq \beta$, then clearly $k \in \rho_i(m_i(a))$, and if $k = \beta$ then $\alpha \in m_i(a)$ because $m_i^{-1}(\alpha) = m_i'^{-1}(\beta)$. So k is in $\rho_i(m_i(a))$, and we have the second inclusion $m_i'(a) \subseteq \rho_i(m_i(a))$. In conclusion, $m_i \equiv m_i'$.

With the property (H_i) at hand, we are now able to prove the correctness of our algorithm. If the output is YES, then if we denote by T and S the last sets computed at step (1), it is obvious that the corresponding models m and m' are equivalent, and so m_u and m_u' are equivalent. In the other hand, if the output is NO, then it is easy to see that the last models m and m' are mot equivalent, and so that m_u and m_u' are not equivalent. □

Thus, in order to determine whether two databases D and D' are equivalent, it is enough to construct their update models, say m_u and m_u', using Algorithm 3.2, and then determine wether $m_u \equiv m_u'$, using Algorithm 4.1.

Let M_u be the set of all the update models associated with the databases of BASES(Σ). Let $M_u/\equiv$ be the set of equivalence classes of update models. It follows from Theorem 4.1, that there is a one-to-one correspondence between the sets BASES(Σ)/≡ and $M_u/\equiv$. We shall come back to this remark in Section 6, when discussing updates.

Having defined our data model, semantic implication in it, and equivalence, we turn our attention now to query and update processing. Given a database, our general plan works as follows:

(1) Express the query or update in terms of the database.
(2) Produce an appropriate model of the database.
(3) Process the query or update in the model.
(4) Give the result in terms of the databse.

In this way, the syntactic component, namely the relational database, serves as an interface in which the user interacts with the system, while the semantic component, namely the model of the database, serves as the environment in which the actual processing takes place.

5. QUERY ANSWERING

In this section we discuss query answering. Throughout our discussions we assume a universe of attributes U, and a consistent database $D = (\delta, \Sigma)$ over U.

In the relational model, a query is a well-formed expression whose operands are relation schemes over U, and whose operations are projection, selection, join, and set theoretic operations [12]. In order to compute the answer to a query, we substitute database relations for corresponding schemes in the expression, and we perform the operations. The basic problem with this definition is that the user is required to know how attributes are grouped together into relation schemes. To avoid this problem, there are now a number of attempts to use a universal relation as the basis of a query language, for example System U [8]. In such a language, although the actual database is a multi-relation database, the user is aware *only* of the universe of attributes. This universe is the interface through which the user interacts with the database as follows:

Queries: the user submits a set Q of attributes and a selection condition and the system returns the set of tuples over Q "implied" by the database and satisfying the given condition.

Updates: the user submits the tuple(s) to be inserted or deleted and the system returns an acknowledgement informing the user whether the update is acceptable (and, therefore, performed). However, ambiguities arise, as the system must "navigate" through relation schemes in order to answer queries or to process updates. These ambiguities come from the purely *syntactic* approach to query answering and update processing in the relational model. In this section, we consider query answering through a universal relation interface. Update processing is discussed in the following section.

5.1. SEMANTIC APPROACH : ANSWERING THROUGH QUERY MODELS

In our investigations, while keeping the philosophy of the universal relation assumption, we avoid ambiguities by adopting a *semantic* approach to query answering. That is, again, the user submits a set of attributes Q and a selection condition. But now, the answer is defined semantically as follows:

(1) Find the set of all tuples over Q *implied* by D, and
(2) return those tuples that *satisfy* the selection condition.

In what follows, we discuss these two steps separately, neglecting (for the moment) questions of efficiency.

Given a relation scheme Q, we denote by $\alpha(Q, D)$ the set of all tuples over Q implied by D, that is

$$\alpha(Q, D) = \{ t \in dom(Q) \mid D \models t \} = \{ t \in dom(Q) \mid t \in T(D) \} = T(D) \cap dom(Q)$$

Recall that $T(D)$ is the set of tuples in TUPLES that are true in every model of D. In principle, in order to compute $\alpha(Q, D)$, for all Q, it is sufficient to

(a) compute $T(D)$ once, and
(b) given a Q, find all tuples in $T(D)$ that are defined over Q.

We have seen in Section 3 that, in order to compute T(D), it is enough to construct a query model of D. Algorithm 3.1 does just that. Let us recall that if m_q is a query model of D, then $T(D) = T(m_q)$. It follows that tuple t is in T(D) if and only if t is true in m_q. Thus, in order to find $\alpha(Q, D)$, it is enough to check which tuples over Q are true in m_q. Let us see an example. Consider the query ABD, in Figure 3.1. In order to answer this query in the database $D_1 = (\delta, \Sigma_1)$, we use the query model

m_{q_1} and we check which tuples over ABD are true in m_{q_1}. We find that $m_{q_1}(a_1b_1d_1) = 3$, $m_{q_1}(a_3b_2d_1) = 4$, and $m_{q_1}(x) = \phi$ for all $x \neq a_1b_1d_1$, $a_3b_2d_1$. Thus

$$\alpha(ABD\ ,\ D_1) = \{\ a_1b_1d_1\ ,\ a_3b_2d_1\ \}$$

To compute the answer of the *same* query in the database $D_2 = (\delta\ ,\ \Sigma_2)$, we must use the query model m_{q_2}. We find now

$$\alpha(ABD\ ,\ D_2) = \{a_1b_1d_1\ ,\ a_3b_2d_1\ ,\ a_2b_2d_1\ \}$$

Notice how a change in the set of dependencies influences the answer, for the *same* set of tuples in the database.

Let us recall now our original plan for answering queries in a database D: the user submits a set of attributes Q and a selection condition, and the system returns all tuples over Q that are implied by D and satisfy the selction condition. We have seen so far an algorithm for producing the set $\alpha(Q\ ,\ D)$ of all tuples over Q implied by D. The next question is how do we select those tuples of $\alpha(Q\ ,\ D)$ that satisfy the selection condition.

We call *elementary condition* over U any expression of the form X = x, where X is a relation scheme over U and x is in dom(X). In the universe of Figure 3.1, $B = b_2$, $AD = a_3d_1$, $C = c_5$, are examples of elementary conditions. We say that a tuple t *satisfies* the elementary condition X = x if and only if $t \models x$. We call *elementary query* over U any set of attributes Q together with an elementary condition ε=(X=x) such that $X \subseteq Q$; we denote it by Q/ε. Given an elementary query Q/ε and a database D, we call *answer* of Q/ε in D, denoted by $\alpha(Q/\varepsilon\ ,\ D)$, the set of tuples in $\alpha(Q\ ,\ D)$ that satisfy ε. That is,

$$\begin{aligned}\alpha(Q/X = x\ ,\ D) &= \{t \in \alpha(Q\ ,\ D)\ /\ t \models x\ \} \\ &= \{\ t \in dom(Q)\ /\ D \models t\ \text{ and }\ t \models x\ \}.\end{aligned}$$

Let us recall that $D \models t$ iff t is true in m_q, where m_q is a query model of D. Therefore, in order to compute $\alpha(Q/\varepsilon\ ,\ D)$, we can use the query model m_q produced by Algorithm 3.1. Indeed, let $\varepsilon = (X=x)$ and assume $X=A_1\ A_2...A_k$ and $x=a_1a_2\cdots a_k$. Starting with m_q define a model $m_{q/\varepsilon}$ as follows:

(1) $m_{q/\varepsilon}(a_i) = m_q(a_i)$, i=1,2, ... ,k

(2) $m_{q/\varepsilon}(a'_i) = \phi$ for all $a'_i \in dom(A_i)$ such that $a'_i \neq a_i$, i=1,2...,k

(3) $m_{q/\varepsilon}(a) = m_q(a)$, otherwise.

In other words, $m_{q/\varepsilon}$ is obtained from m_q by setting to empty every symbol of $dom(A_i)$ different than a_i, i=1,2,...,k. Clearly, a tuple t is in $\alpha(Q/\varepsilon, D)$ if and only if t is true in $m_{q/\varepsilon}$. For example, in Figure 3.1, assuming the elementary condition $\varepsilon = (A = a_2)$, we obtain $m_{q/\varepsilon}$ by setting a_1 and a_3 to empty, in m_{q_1}. Here are some examples of computations in Figure 3.1, assuming $\varepsilon = (A = a_2)$:

In $m_{q1/\varepsilon}$: $\alpha(AB/\varepsilon, D_1) = \{ a_2b_2 \}$, $\alpha(AD/\varepsilon, D_1) = \phi$,
In $m_{q2/\varepsilon}$: $\alpha(AB/\varepsilon, D_2) = \{ a_2b_2 \}$, $\alpha(AD/\varepsilon, D_2) = \{ a_2d_1 \}$.

Elementary conditions can be combined to form more complex conditions. We call *selection condition* over U any well formed expression whose operands are elementary conditions and whose operations are negation, conjunction and disjunction. In the example of Figure 3.1, the following is a selection condition :

$$s = (\neg(B = b_1)) \wedge ((AD = a_2d_1) \vee (C = c_1))$$

It is customary to write $X \neq x$ instead of $\neg(X = x)$. The following definition summarizes our discussion so far, by stating formally what we mean by a query and its answer.

Definition 5.1.1 Let U be a universe. A *query* over U is any set of attributes Q together with a selection condition s; it is denoted by Q/s. Given a query Q/s and a database D over U, the *answer* of Q/s in D, denoted by $\alpha(Q/s, D)$, is defined recursively as follows: for any selection conditions s and s',

(1) $\alpha(Q/\varepsilon) = \{t \in \alpha(Q, D) / t \models x\}$ where ε is (X=x)
(2) $\alpha(Q/\neg s, D) = \alpha(Q, D) - \alpha(Q/s, D)$
(3) $\alpha(Q/s \wedge s', D) = \alpha(Q/s, D) \cap \alpha(Q'/s, D)$
(4) $\alpha(Q/s \vee s', D) = \alpha(Q/s, D) \cup \alpha(Q'/s, D)$. □

In order to compute the answer to $\alpha(Q/s, D)$, we compute recursively a model $m_{q/s}$ as follows:

(1) $m_{q/s} = m_{q/\varepsilon}$, if s is an elementary condition ε.
(2) $m_{q/\neg s}(a) = m_q(a) - m_{q/s}(a)$,
(3) $m_{q/s \wedge s'}(a) = m_{q/s}(a) \cap m_{q/s'}(a)$,
(4) $m_{q/s \vee s'}(a) = m_{q/s}(a) \cup m_{q/s'}(a)$, for all a in SYMBOLS.

We clearly have the following property:

for any query Q and selection condition s, we have $\alpha(Q/s, D) = T(m_{q/s})$ □

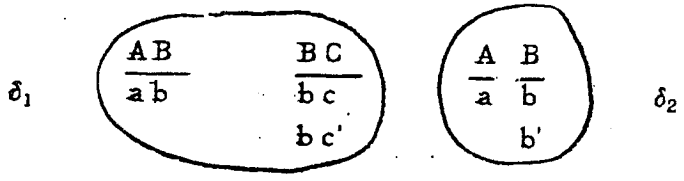
For example, in order to compute the answer of $\alpha(AD/ -(A=a_1)\ ,\ D_2)$ in the database D_2 of Figure 3.1, we compute the model $m_{q_2/-(A=a_1)}$ from the model m_{q_2} of Figure 3.1. We find :

$m_{q_2/s}$: $a_1 \rightarrow \phi$, $a_2 \rightarrow 2$, $a_3 \rightarrow 4$, $b_1 \rightarrow 13$, $b_2 \rightarrow 24$, $c_1 \rightarrow 1234$, $d_1 \rightarrow 1234$,

and so the answer to the query is:

$T(m_{q_2/s}) = \{\ a_2d_1\ ,\ a_3d_1\ \}$. ▫

We have seen so far that, while keeping the philosophy of the universal relation assumption, we can avoid ambiguities by adopting a purely semantic definition of answers. We have also seen an algorithm for *deductive* query answering. One may wonder where exaclty relational joins fit into the picture. Let us see some examples. Consider the databases $D_1 = (\delta_1\ ,\ \Sigma_1)$ and $D_2 = (\delta_2\ ,\ \Sigma_2)$, shown below, where we assume that Σ_1 and Σ_2 are empty.



First, let us consider the query Q_1 = ABC in the database D_1. From what we have said so far, it is clear that the answer to Q_1 is empty, that is, $\alpha(Q_1\ ,\ D_1) = \phi$. On the other hand, the join of the two relations in δ_1 produces the tuples abc and abc'. Although these tuples are not true in *every* model of D_1, there is a model of D_1 in which abc is true, and there is a model of D_1 in which abc' is true (actually, there is even a model of D_1 in which *both* tuples are true). Consider next the query Q_2 = AB in the database D_2. Clearly, $\alpha(AB\ ,\ D_2) = \phi$. On the other hand, the join of the two relations in δ_2 produces the tuples ab and ab'. Again, although these tuples are not true in *every* model of D_2, there is a model of D_2 in which ab is true, and there is a model of D_2 in which ab' is true. These observations suggest the definition of an upper bound for query answers, that we shall now define.

Let D be a database and m a model of D. Let us recall that T(m) denotes the set of all tuples that are true in m, and that $T(D)$ denotes the set of all tuples that are true in *every* model of D, that is,

(1) $T(D) = \bigcap \{\ T(m)\ /\ m \text{ is a model of } D\ \}$.

Let us also recall that, given a query Q, the answer $\alpha(Q\ ,\ D)$ is the set of tuples over Q that are true in every model of D, that is,

(2) $\alpha(Q, D) = dom(Q) \cap T(D)$.

Now, let us denote by SYMBOLS(D) the set of symbols in SYMBOLS that are assigned a non-empty set by every model of D. Clearly, SYMBOLS(D) is the set of the symbols appearing in the database. Let us then denote by TUPLES(D), the set of all tuples over SYMBOLS(D). Now, let us denote by $T^*(D)$ the set of all tuples in TUPLES(D) such that, there is a model of D in which t is true. Clearly, we have:

(1*) $T^*(D) = \bigcup \{ T(m) \cap TUPLES(D) \ / \ m \text{ is a model of } D \}$.

Given a query Q, let us denote by $\alpha^*(Q, D)$ the set of all tuples t over Q such that, there is a model of D in which t is true. Clearly, we have:

(2*) $\alpha^*(Q, D) = dom(Q) \cap T^*(D)$.

It is not difficult to see that $T(D) \subseteq T^*(D)$ and $\alpha(Q, D) \subseteq \alpha^*(Q, D)$. Now, given a database $D = (\delta, \Sigma)$, two questions arise:

(a) How can we compute $T^*(D)$
(b) Under what conditions on Σ, $T(D) = T^*(D)$

For more details on these and other related questions, the reader is refered to [10]. Let us only mention here that $T^*(D)$ can be computed by slightly modifying the algorithm 3.1, seen earlier. It turns out that the output of this new algorithm is precisely what we would get from δ using joins (and projections). This result and the fact that $\alpha(Q, D) \subseteq \alpha^*(Q, D)$ show that joins can be seen as producing crude upper bounds of answers.

5.2. SYNTACTIC APPROACH : ANSWERING THROUGH TABLES.

As we have said in the introduction, the model used in this paper is a recasting of the partition model proposed in [10]. The partition model is used in [3] in order to justify an assumption commonly used in database theory, namely the weak-instance assumption. In this section, we study the relationship between our model and the weak-instance assumption of the relational model. In particular, we show that the well-known chase algorithm which is used to determine consistency of a database can, in fact, be used to compute query answers, in the sense defined earlier (see Definition 5.1.1). First, let us recall some definitions.

Definition 5.2.1 Let U be a universe and V a countable set of symbols not in SYMBOLS. The symbols of V are called *variables*. A *table* over U and V is a set τ of tuples over U such that

(1) $\forall t \in \tau$, $\forall A \in U$, $t(A) \in$ dom(A) or t(A) $\in V$

(2) $\forall t, t' \in \tau$, $\forall A,B \in U$, $t(A) \in V \wedge t'(B) \in V \implies t(A) \neq t'(B)$.

We denote by T(U,V) the set of all tables over U and V. □

Roughly speaking, the tuples of a table can contain constantss (i.e. elements of the attributes domains) as well as variables (i.e. elements of V). The constants in a tuple form a subtuple of particular interest, that we now define.

Definition 5.2.2 Let τ be a table over U and V, and let t be a tuple of τ. The *content* of t is a tuple denoted by $|t|$ and defined as follows:

(1) $|t|$ is a tuple over the set { A / t(A) $\notin$ V },

(2) For every A in U such that t(A)$\notin$V, $|t|$(A) = t(A).

We denote by $|\tau|$ the set { $|t|$ / t $\in \tau$ }. □

Tuples having the same content can be seen as "equivalent", and this definition can be extended to tables.

Definition 5.2.3 Two tables τ and τ' are called *equivalent*, denoted by $\tau \equiv \tau'$, if and only if $|\tau| = |\tau'|$. □

It follows that τ and τ' are equivalent if they are equal, up to a renaming of variables. We denote by $\overline{\tau}$ the equivalence class of τ and by T(U,V)/$\equiv$ the set of all eqquivalence classes in T(U,V).

Let I(U) be the set of all interpretations of U. As attribute domaains are countably infinite, the set TUPLES is also countably infinite, and there is a bijection between TUPLES and the set of (positive) integers ω. Let ε be such a bijection. Now, using ε, let us define a function Ψ from T(U,V)/$\equiv$ into I(U) as follows: for all τ in T(U,V), $\Psi(\overline{\tau})$ is an interpretation I in I(U), such that:

$\forall a \in$ SYMBOLS, I(a) = { $\varepsilon(|r|)$ / $r \in \tau \wedge |r|$(A) = a }.

Clearly, $\Psi(\overline{\tau})$ does not depend on the representative τ chosen.

Lemma 5.2.1 For all s in TUPLES and for all τ in T(U,V), we have:

s $\in$ T($\Psi(\overline{\tau})$) $\iff$ $\exists t \in \tau$, s is a subtuple of $|t|$.

(Recall that T($\Psi(\overline{\tau})$) denotes the set of all tuples that are true in the interpretation $\Psi(\overline{\tau})$). □

Proof

omitted □

Algorithm 5.2.1 (CHASE).

Input : a table τ in T(U,V).

Output : a table $chase_\Sigma(\tau)$.

Do the following until there is no more change:

·*if* there is r and s in τ, X→A in Σ such that:

r(X) = s(X) and r(A) ≠ S(A), and r(A)∉V and s(A)∈V

then change s(A) to r(A). □

The chase algorithm has been extensively studied and used in the relational literature [12]. However, this algorithm has been almost exclusively used to test consistency of a database with repect to a set of dependencies Σ. In what follows, we show that $chase_\Sigma(\tau)$ can also be used to answer queries, and that the answers thus obtained are precisely those obtained by the semantic approach described earlier (Section 5.1). The chase algorithm given here, uses the background assumption that the set Σ is closed under implication by the condition "r(A)∉V ". This chase algorithm would not be correct if the set of functional dependencies Σ was not closed under implication.

Consider a database $D=(\delta, \Sigma)$ over a universe U, and associate with δ a table $\tau(\delta)$ such that $|\tau(\delta)| = \{ t \in \delta(R) \ / \ R \subseteq U, R \neq \phi \}$. Clearly, this definition specifies only the constants of the table $\tau(\delta)$ but not the variables. It follows that δ is actually associated with the equivalence class of $\tau(\delta)$. Now, $chase_\Sigma$ can be seen as a function from T(U,V) into itself. Moreover, this function preserves equivalence, that is, if $\tau \equiv \tau'$, then $chase_\Sigma(\tau) \equiv chase_\Sigma(\tau')$. (This is clear from the chase algorithm.) Thus, we can define a function $chase_\Sigma$ from $T(U,V)/\equiv$ into itself as follows: $chase_\Sigma(\overline{\tau}) = \overline{chase_\Sigma(\tau)}$. The following theorem describes how semantic implication can be tested through syntactic manipulations of tables.

Theorem 5.2.1 For every database (δ, Σ) and every tuple t in TUPLES,

$t \in T((\delta, \Sigma)) \iff \exists s \in |chase_\Sigma(\overline{\tau(\delta)})|$, t is a subtuple of s. □

Proof

omitted □

Consider now a database $D=(\delta, \Sigma)$ and the associated $chase_\Sigma(\tau(\delta))$. Let Q be any query (without selection condition). It follows from Theorem 5.2.1 that, in order to compute the answer $\alpha(Q, D)$, it is enough to perform the following operations:

(1) Project $chase_\Sigma(\tau(\delta))$ over the attributes of Q.

(2) From the result of (1), select all tuples that are total (a tuple is total if it is equal to its content).

In the general case, where the query consists of a set of attributes Q and a selection condition s defined as in Section 5.1, the answer $\alpha(Q/s, D)$ is now computed as follows:

(1) Select from the tuples of $chase_\Sigma(\tau(\delta))$ those that satisfy s (in the relational sense).

(2) Project the result of (1) over the attributes of Q

(3) From the result of (2), select all tuples that are total.

This can also be expressed by the following formula of relational algebra:

$$\alpha(Q/s, D) = \Pi\downarrow_Q (\sigma_s (chase_\Sigma(\tau(\delta))))$$

where $\Pi\downarrow_Q(r)$ is the set of all total tuples in $\Pi_Q(r)$.

In conclusion, we have established in this section the relationship between our semantic approach to query answering and the syntactic approach of the relational model. In fact, Theorem 5.2.1 can be seen as providing a semantic content to the weak-instance assumption.

6. UPDATING

In this section, we discuss database updating, and we give an algorithm for performing the insertion or deletion of a tuple in a given database. We begin by defining formally insertion and deletion of a tuple. In doing so, we shall require that the insertion or deletion of a tuple produces a "minimal change" in the original database. We consider this to be a reasonable constraint, from a practical viewpoint.

Let U be a universe, and Σ a set of functional dependencies over U. recall that $BASES(\Sigma)$ denotes the set of all consistent databases over U having Σ as their set of dependencies. Given a database $D=(\delta, \Sigma)$ and any tuple t in TUPLES, we define the result of the insertion of t in D as a database D' verifying the following properties:

(1) $D \prec D'$

(2) $D' \not\models t$

(3) for every database D'' verifying (1) and (2), we have $D' \prec D''$.

It is clear that the result of the insertion of t in D does not always exist, and, when it is defined, it is not unique. But it is also clear that two databases verifying (1), (2), and (3) are equivalent. So we can define a (partial) function $\overline{INS}$ from BASES(Σ)/≡ x TUPLES into BASES(Σ)/≡ as follows:

Definition 6.1 Given a database D in BASES(Σ) and a tuple t in TUPLES, the *insertion of t in* $\overline{D}$, denoted by $\overline{INS}(\overline{D},t)$ is the minimal class $\overline{D}'$ of BASES(Σ)/≡ (if such a class exists) such that

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

(2) $D' \models t$.

We have seen in Section 4, that two equivalence databases cannot be distinguished by the user in a universal relation interface. So, the differences between elements of an equivalent class of BASES(Σ)/≡ are immaterial for the user. This is why we choose to discuss here updating of equivalence classes rather than of particular databases. We shall come back to this remark at the end of the section. We now define deletion of a tuple t in a database D, in much the same manner as we have done for insertions.

Definition 6.2 Given a database D in BASES(Σ) and a tuple t in TUPLES, *the deletion of t in* $\overline{D}$, denoted by $\overline{DEL}(\overline{D},t)$ is the maximal class $\overline{D}'$ of BASES(Σ)/≡ verifying

(1) $\overline{D}' \leqslant \overline{D}$,

(2) $D' \not\models x$, *for all* x *such that* $x \models_D t$

The following lemma gives a characterisation of $\overline{DEL}(\overline{D},t)$.

Lemma 6.1 Let D be a database in BASES(Σ) and t a tuple in TUPLES. Then

$T(\overline{DEL}(\overline{D},t)) = T(D) - \{x \in T(D) , x \models_D t\}$ ▫

Proof easy ▫

An important consequence of this lemma is that $\overline{DEL}$ is always defined, for every value of D or t. Let us put these definitions to work in the following example.

Example 6.1 Consider the database D of Figure 3.3. We have T(D)={a,b,c,b',c',ab,bc,b'c',abc,ac}. If we insert the tuple t=a'b' in D, we obtain from Definition 6.1 :

$T(\overline{INS}(\overline{D},a'b'))$ = {a,a',b,b',c,c',ab,bc,ac,abc,a'b',b'c',a'c',a'b'c'}.

The tuples a'b'c' and a'c' have also been inserted because of the dependency B→C in Σ. We have a'b' $\models_D$ a'b'c' and a'b'c' $\models_D$ a'c', that is, a'b'c' and a'c' are "consequences" of a'b'. If we now want to delete from D the tuple t=abc, then we obtain

from Definition 6.2:

$T(\overline{DEL}(\bar{D},abc) = \{a,b,bc,b'c',ac\}$.

All the tuples from which abc is a consequence, namely abc and ab, have been removed from T(D). □

We now consider the following problem. For a given database D and a tuple t, find algorithms to process the insertion or the deletion of t in $\bar{D}$. We use the notion of update model introduced in Section 3. We have seen that an upddte model characterizes exactly one equivalence class of BASES(Σ)/≡. So, in order to process an insertion or a deletion, all we have to do is to compute two mappings i and d transforming the update model of $\bar{D}$ into the update model of $\overline{INS}(\bar{D},t)$ or $\overline{DEL}(\bar{D},t)$, respectively. We can represent the situation in the following figure.

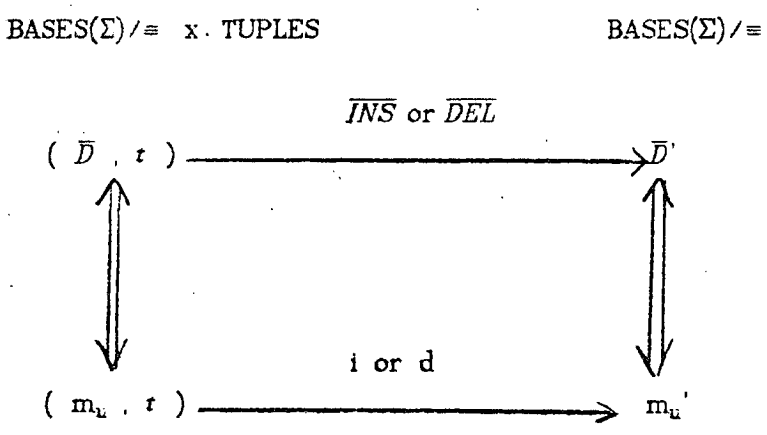


Figure 6.1 Functions i and d, "translating" $\overline{INS}$ and $\overline{DEL}$, respectively.

So the update processing consists of the following steps

(1) Express the update in terms of the database.
(2) Produce an update model of the database.
(3) Process the update in this model.
(4) Given the result in terms of the database.

Notice that these steps are very similar to those used for query processing in the previous section. The following algorithm computes the function i.

Algorithm 6.1 Insertion.

Input : An update model m_u and a tuple t.

Output: An update model m_u'.

(0) Let m_1 be the model m_u'.

(1) For every sub-tuple $s=a_1 \cdots a_k$ of t, if $m_u(s) = \phi$,
add a new integer i_s in $m_u(a_j)$ for j in {1,...,k}.

(2) -For $j \geq 1$, compute m_{j+1} from m_j as follows, until $m_{j+1} = m_j$:
If there is $X \rightarrow A \in \Sigma$, $x \in dom(X)$ and $a \in dom(A)$ such that
$m_j(x) \cap m_j(a) \neq \phi$ and $m_j(x) \not\subseteq m_j(a)$,
then
- compute m_j' as follows:
$m_j'(a) = m_j(a) \cup m_j(x)$,
$m_j'(b) = m_j(b)$, for every b in SYMBOLS such that $b \neq a$, and
- m_{j+1} is the interpretation obtained from m_j' as follows:
For every tuple $t=a_1 \cdots a_k$ such that $t\, m_j'(t) \neq \phi$ and $m_j(t) \neq \phi$,
add a new integer i_t to $m_j'(a_p)$ for all p in {1...k}.
else let m_{j+1} be m_j.

(3) -Let m_{i_0} be the last function computed at Step 2. Let m_u be
the irreducible model obtained by removing from m_{i_0} an element of
each pair of inseparable integers. ▫

Theorem 6.1 Algorithm 6.1 terminates, and if the function m_u' thus obtained is an interpretation of U, then $i(\overline{D},t) = m_u'$ (up to a renaming of the integers) else $\overline{INS}(\overline{D},t)$ is not defined. ▫

Proof The proof of this theorem is quite similar to the proof of Theorem 3.3 ▫

Algorithm 6.1 strongly resembles to Algorithm 3.2, and this is not surprising. Indeed, the construction of an update model for a database D can be seen as the insertion of all the tuples of D into the empty database.

We give now an algorithm for computing the function d.

Algorithm 6.2 Deletions.

Input : an update model m_u and a tuple t.

Output: an update model m_u'.

The update model m_u' is obtained from m_u by removing from $m_u(a)$ the integers of $m_u(t)$ □

Theorem 6.2 The model m_u' computed by Algorithm 6.2 is the update model $d(m_u,t)$ (up to a renaming of the integers). □

Proof

We clearly obtain an update model from the algorithm. We can use Lemma 3.1 to characterize the elements of the input model m_u. We obtain, using the notations of Section 3,

(1) $m_u'(t) = \{ k_s , s \models_D t \}$

We clearly also have

(2) $s \models_D t$ iff $m_u(s) \subseteq m_u(t)$.

In conclusion, condition (2) implies that it is necessary and sufficient to remove from m_u all the integers of $m_u(t)$ in order to eliminate from T(D) the tuples s such that $s \models_D t$, and condition (1) shows that we do not remove any thing else from T(D). □

We have seen, so far, that we can express updates and then process them using our semantic approach. We have considered only one-tuple updates in order to simplify the presentation. All the results and algorithms can obviously be extended to multi-tuple updates. We conclude this section by a brief discussion of how to represent the classes $\overline{INS}(\overline{D},t)$ or $\overline{DEL}(\overline{D},t)$ by an appropriate database D'. That is, how can we define two functions INS and DEL from BASES(Σ) x TUPLES into BASES(Σ) verifying the following properties: for example, if D' = INS(D,t)

(1) $\overline{D}' = \overline{INS}(\overline{D},t)$

(2) $\forall Q \subseteq U , \forall t \in \delta(Q) , t \in T(D') \implies t \in \delta'(Q)$.

(3) δ' is minimal with repect to inclusion.

Property (2) says that every tuple t which is in $\delta(Q)$ and which is still true in D' must appear in δ'. Property (3) says that D' must not be redundant.

The representation problem just described has been traditionally considered in the relational literature as *the* update problem [12]. Obviously, it is an important problem from the system viewpoint. However, in a universal relation interface, and from the user viewpoint, it is immaterial which representation is used. What really matters is the equivalence class. This aspect of the update problem has been largely ignored in the relational literature. This is why we have chosen in this section to discuss updating of equivalence classes rather than of particular databases. We believe that both aspects are important, but we also believe that working with

equivalence classes provides deeper information on update semantics.

7. CONCLUSION

We have seen a set-theoretic interpretation of the relational model which adds to it a deductive component through set-containment. While keeping the philosophy of the universal relation assumption, we have avoided ambiguities by adopting a purely semantic approach to query answering. We have also presented algorithms for query answering and update processsing, and for determining equivalence of databases.

We are currently working on several aspects of our approach. Firstly, it has been shown in [10] that set-containment allows to capture the notion of inheritance. Functional dependency is just one example of inheritance. So the next step is to investigate the influence of inheritance properties, in general, on query and update processing. A second important aspect is computational compexity. The number of symbols and the number of tuples in the database have a direct impact on the efficiency of our inference algorithm. However, it seems that keys (in the sense defined in the relational model) can be profitably used to increase efficiency. Finally, we are looking into the problem of recursively defined queries. Our definition of an interpretation does not allow us to model databases where two or more attributes have a common domain. For example, the relation {ab , bc} can never have a model. Indeed, as a and b are symbols of the same domain, we have: $I(a) \cap I(b) = \phi$, for every interpretation I. It follows that no interpretation can ever verify the tuple ab. However, there is a way out through a modest extension in the definition of an interpretation [11]. Indeed, it is sufficient to require that, for every symbol a, there are symbols a' and a'' (in the same domain) such that $I(a) \subseteq I(a') \cap I(a'')$. The resulting theory and computational algorithms are presented in a forthcoming report [11].

Bibliography :

[1] Beeri C., Mendelzon A.O., Sagiv Y., Ullman J.D., Equivalence of Relational Database Schemes, SIAM J. on Computing 10:2, May 1981, 352-370.

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

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

[4] Hull R., King R., Semantic Database Modeling: Survey, Applications, and Research Issues, U.S.C. Computer Science Department Technical Report TR-86-201.

[5] Fagin R., Horn Clauses and Database Dependencies, ACM Symposium on Theory of Computing 1982 123-134.

[6] Gallaire H., Minker J., Nicolas J.M., Logic and Databases : A Deductive Approach, Proceedings ACM-PODS, 1983.

[7] Imielinski T., Lipski W., Incomplete Information in Relational Databases. J.ACM 31,4, October 1984, 761-791.

[8] Korth H., System/U: Progress Report, XP2 Workshop on Relational Database Theory, Pennsylvania State Univ., June 1981.

[9] Reiter R., Towards a Logical Reconstruction of Relational Database Theory, in Conceptual Modeling : Perpectives from Artificial Intelligence, Databases and Programming Languages, Springer-Verlag, to appear.

[10] Spyratos N., The Partition Model: A Deductive Database Model, INRIA Research Report no 286, April 1984 (also to appear in ACM-TODS).

[11] Spyratos N., Lecluse C., A Logic For Relational Databases, INRIA Research Report, in preparation.

[12] Ullman J.D., Principles of Database Systems, Rockville, MD:Computer Science Press.

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique