



# 1

---

## UNCERTAINTY IN INTELLIGENT DATABASES

Robert Demolombe

*ONERA/CERT  
Toulouse, France*

### 1 INTRODUCTION

By an *intelligent database* we mean a traditional relational database with additional functionalities to represent either (1) general rules, as in deductive databases, or (2) some kind of incomplete information, like marked null values or disjunctive facts, or (3) additional meta-information, like information validity, uncertainty factors, or some kind of modality.

We shall assume first that such a database can be represented by a first order theory  $DB$ . This theory is intended to represent a part of the world, called  $W$ , that is formally represented by an interpretation of the same language used to represent the theory  $DB$ . Subsequently, we shall see that first order theories are not enough to represent certain kinds of additional information.

The subject of uncertainty in intelligent databases is broad and covers many different approaches and interpretations. The following discussion distinguishes among several different kinds of uncertainty.

#### *Incompleteness*<sup>1</sup>

The first kind of uncertainty refers to a situation where we know that  $A \vee B$  holds, but we are uncertain which of the two ( $A$  or  $B$ ) holds. For example, we may know that “John teaches Mathematics or Logic”, represented by

---

<sup>1</sup>In this chapter the terms *incompleteness* and *validity* are used in the sense defined by Motro in [22, 23]. They do not refer to properties of a formal system about the links between syntax and semantics, as is the case in logic.

$Teach(John, Mathematics) \vee Teach(John, Logic)$ , but not which of these courses he actually teaches. The uncertainty is about what is true in the world.

More generally, incompleteness is a form of uncertainty in the sense that if the theory  $DB$  has several distinct (up to an isomorphism) models  $M_1, M_2, \dots, M_n$ , we are not certain whether  $W$  is (up to an isomorphism)  $M_1$ , or  $M_2$ , or  $\dots$  or  $M_n$ . This can be expressed by

$$W = M_1 \quad \text{or} \quad W = M_2 \quad \text{or} \quad \dots \quad \text{or} \quad W = M_n$$

### *Validity*

Even if the information  $A$ , say  $Teach(John, ComputerScience)$ , is in  $DB$ , we are not always certain that  $A$  holds in the world. In this case uncertainty is about whether  $A$  is true or false in  $W$ . This type of uncertainty refers to information validity.

This notion of uncertainty is unrelated to the former one, and it can be defined whether or not  $DB$  is a complete theory. If we call  $M(DB)$  the set of  $DB$  models, it expresses lack of knowledge about whether  $W$  is a model of  $DB$  or not. This can be expressed by

$$W \in M(DB) \quad \text{or} \quad W \notin M(DB)$$

### *Inconsistency*

If inconsistent data like  $A$  and  $\neg A$  can be derived from a database  $DB$ , in most cases it is because these data have been inserted by different agents that have different beliefs about the world. For example, in the context of distributed databases, if each database is considered an agent, it may happen that  $A$  is derivable from the database  $DB_1$ , and  $\neg A$  is derivable from the database  $DB_2$ .

For example,  $DB_1$  might have  $\{Male(Jane), (\leftarrow Male(x) \wedge Female(x)) \forall x\}$ , and therefore  $\neg Female(Jane)$  is derivable from  $DB_1$ , whereas  $DB_2$  might have  $Female(Jane)$ . In this case uncertainty is about the database that contains the correct information.

The mutual inconsistency of  $DB_1$  and  $DB_2$ , that is, their lack of common model, can be expressed by

$$W \in M(DB_1) \quad \text{or} \quad W \in M(DB_2)$$

## *Vagueness*

A predicate used to represent information in a database might not have a precise definition, with the result that we are not certain whether or not a given object satisfies the predicate. This uncertainty does not come from lack of information about the world; it comes from the vagueness of the language itself.

If we consider, for example, the predicate  $Tall(x)$ , and the person John, it may be uncertain whether John is a tall person. In this situation one can assign a value in  $[0, 1]$ , say 0.75, to a fact such as  $Tall(John)$ . The intuitive interpretation of such a value is somewhat controversial, as we can propose at least two interpretations.

The first interpretation is that the value represents a distance between the properties of a perfectly tall person, and John's properties. However, in many cases, the perfect properties are not explicitly defined, and there is no definition of a distance measure over these properties; consider, for example, the vague predicate "good restaurant".

Another interpretation is that each agent has his own definition of what is a tall person. That is, each agent can define the extension of the predicate  $Tall(x)$ , but these extensions are not the same for all the agents. In other words, each agent has a precise definition of the predicate, but there is no consensus about this definition. In this case the coefficient 0.75 can be interpreted as the percentage of agents that agree on the fact that  $Tall(John)$  is true.

This chapter surveys only the first two kinds of uncertainty: *incompleteness* and *validity*. We shall try to reformulate in a logical framework the different formalisms presented by the different authors. At times, we shall change some aspects of the original works, for the sake of a simplified and uniform presentation. We hope that this would not significantly alter the intentions of the authors. Our survey is presented according to the kind of uncertainty; for each kind we consider how information is represented, techniques for reasoning with this information in order to compute answers to queries, the sources of the uncertainty, and the kind of constraints that can be defined to limit uncertainty. Our survey discusses representative works in these areas, and it does not attempt to be exhaustive.

## 2 INCOMPLETENESS

### 2.1 Representation of Uncertainty

We consider here how incompleteness is represented, starting from standard relational databases, and ending with more sophisticated types of intelligent databases.

#### *Relational Databases*

In standard relational databases information is represented by a set of tables. Each table contains a set of facts of one predicate. The column names, called *attributes*, allow easy reference to the predicate arguments. As an example, consider the table *Teach*

| <i>Teach</i>   |                    |
|----------------|--------------------|
| <i>Teacher</i> | <i>Course</i>      |
| <i>John</i>    | <i>Mathematics</i> |
| <i>Peter</i>   | <i>Logic</i>       |

The associated first order theory  $DB_1$  is quite simple. It contains as many ground facts as there are lines in the table:

$$DB_1 = \{Teach(John, Mathematics), Teach(Peter, Logic)\}$$

The language associated with this theory is formed with the predicate  $Teach(x, y)$ , and the set of constants  $\{John, Peter, Mathematics, Logic\}$ .

An implicit assumption in relational databases is that each constant name refers to a distinct element. This assumption is represented in the theory by *Unique Name Axioms (UNA)*, as defined by Reiter [27]. In this section we shall assume that the theory associated with a database always contains unique name axioms, although, for simplicity, they will not be specified explicitly.

The standard definitions of the relational model also implicitly assume that the domain of each predicate argument is known. This assumption can be represented in logic by a *Domain Closure Axiom (DCA)* as defined by Reiter [27] (see next section). However, in practical situations the definition of these domains is problematic because the set of elements that may occur in a column may be unknown. For example, in the table *Teach*, we know the set of teachers that

appear in that state of the table, but we might not know the set of teachers that may appear in future states. Therefore, in this chapter, the *DB* theories will not contain the DCA.

An interpretation of the language is defined by a domain  $D$  and a meaning function  $m$ , that assigns to each predicate a set of tuples formed with elements of the domain. We assume that to each constant in the language, i.e., to each constant that appears in a table, a distinct element of the domain is assigned, which has the same name as the constant. For example, the constant *John* is interpreted by an element of the domain also named *John*.

Examples of models of the theory *DB* include

- $M_1 = \langle D_1, m_1 \rangle$  where

$$\begin{aligned} D_1 &= \{John, Peter, Mathematics, Logic\} \\ m_1(Teach) &= \{\langle John, Mathematics \rangle, \langle Peter, Logic \rangle\} \end{aligned}$$

- $M_2 = \langle D_2, m_2 \rangle$  where

$$\begin{aligned} D_2 &= \{John, Peter, Mathematics, Logic\} \\ m_2(Teach) &= \{\langle John, Mathematics \rangle, \langle Peter, Logic \rangle, \\ &\quad \langle Peter, Mathematics \rangle\} \end{aligned}$$

- $M_3 = \langle D_3, m_3 \rangle$  where

$$\begin{aligned} D_3 &= \{John, Peter, Mathematics, Logic, Programming\} \\ m_3(Teach) &= \{\langle John, Mathematics \rangle, \langle Peter, Logic \rangle, \\ &\quad \langle John, Programming \rangle\} \end{aligned}$$

In this example, the representation  $W$  of the world might be, for example,  $M_3$ .

Except for the UNA, this logical representation of a relational database corresponds to what Reiter called the *Open World Assumption (OWA)*. This means that there is no assumption about the truth value of facts that are not present in a table. For example,  $\langle Peter, Mathematics \rangle$  is not in the table *Teach*, so we make no assumption about whether or not  $W$  contains  $Teach(Peter, Mathematics)$ .

## Relational Databases with Null Values

Null values have been added to the relational model by Codd [4]. Null values can have many different meanings. The meaning we consider here corresponds

to a value that exists, but is unknown. All the occurrences of null values are represented by the symbol  $-$ , but it is not assumed that all these unknown values are identical. As an example, consider the table

| <i>Teach</i>   |                    |
|----------------|--------------------|
| <i>Teacher</i> | <i>Course</i>      |
| <i>John</i>    | <i>Mathematics</i> |
| <i>Peter</i>   | <i>Logic</i>       |
| <i>Paul</i>    | $-$                |
| $-$            | <i>Programming</i> |

In the associated first order theory null values are represented by existentially quantified variables;

$$DB_2 = \{Teach(John, Mathematics), Teach(Peter, Logic), \\ \exists x Teach(Paul, x), \exists y Teach(y, Programming)\}$$

Null values represent a special kind of uncertainty. For example, if we know the fact  $\exists x Teach(Paul, x)$ , then we know that  $x$  is an element in the extension of  $Teach(Paul, x)$ , but are unaware which element it is.

### *Relational Databases with Marked Null Values*

Marked null values are a refinement of “simple” null values. Reiter [27] and Imielinski and Lipski [15] (and see also [32, 31, 1, 16]) introduced special symbols, called *variables*, to represent marked null values. The same variable may have several occurrences in a database, and each occurrence refers to the same value, though this value is unknown. An example is the table

| <i>Teach</i>   |                    |
|----------------|--------------------|
| <i>Teacher</i> | <i>Course</i>      |
| <i>John</i>    | <i>Mathematics</i> |
| <i>Paul</i>    | $\omega_1$         |
| $\omega_2$     | <i>Logic</i>       |
| $\omega_2$     | <i>Programming</i> |

Such tables are called V-tables in [15]. The associated first order theory is

$$DB_3 = \{Teach(John, Mathematics), \exists x Teach(Paul, x), \\ \exists y (Teach(y, Logic) \wedge Teach(y, Programming))\}$$

In fact, marked null values are, in the logical framework, Skolem constants. The Skolemization of  $DB_3$  leads to

$$DB'_3 = \{Teach(John, Mathematics), Teach(Paul, \omega_1), \\ Teach(\omega_2, Logic), Teach(\omega_2, Programming)\}$$

From a practical point of view, the only difference between these Skolem constants and standard constants is that Skolem constants are not covered by unique name axioms like  $\neg(\omega_1 = Mathematics)$  or  $\neg(\omega_1 = \omega_2)$ .

Nevertheless, Demolombe and Fariñas del Cerro [10] have extended the marked null values approach to represent constraints on marked null values of the form  $\neg(\omega_1 = Mathematics)$ . This extension allows to represent, for example, the information  $\exists x (Teach(Paul, x) \wedge \neg(x = Mathematics))$ .

### Relational Databases with Conditional Tuples

Imielinski and Lipski [15] introduced tables with conditional tuples. To each table, an additional column is added that contains Boolean expressions formed only with the equality predicate. The intuitive interpretation is that a tuple corresponding to a line in the table satisfies the corresponding predicate if, for a given interpretation of the Skolem constants, the Boolean expression takes the value *true*. These tables are called *conditional tables*, or *C-tables* for short. For example,

| <i>Teach</i>   |                    |   |
|----------------|--------------------|---|
| <i>Teacher</i> | <i>Course</i>      | <i>con</i>                                  |
| <i>John</i>    | <i>Mathematics</i> | <i>true</i>                                 |
| <i>Paul</i>    | $\omega_1$         | $\neg(\omega_1 = Mathematics)$              |
| $\omega_2$     | <i>Logic</i>       | $(\omega_2 = Peter) \vee (\omega_2 = Paul)$ |

The information captured in this table, denoted  $Rep(Teach)$ , is a *set* of relations defined in the following way. For each interpretation of the marked null values  $\omega_1$  and  $\omega_2$ , the conditions take the value *true* or *false*, and all the relations that contain the lines for which the condition is true are in  $Rep(Teach)$ .

Notice that, in the interpretation, when  $\omega_2$  is interpreted by *John*, the condition  $(\omega_2 = Peter) \vee (\omega_2 = Paul)$  takes the value *false*. In this case, we notice that (1) relations in  $Rep(Teach)$  may or may not contain the tuple  $\langle John, Logic \rangle$ ,

and (2) there are relations in  $Rep(Teach)$  that contain neither  $\langle Peter, Logic \rangle$  nor  $\langle Paul, Logic \rangle$ ; therefore, the last line in the table cannot be interpreted as  $Teach(Peter, Logic) \vee Teach(Paul, Logic)$ . However, it is possible to represent this disjunctive information with the C-table  $Teach'$ ,

| <i>Teach'</i>  |               |                          |
|----------------|---------------|--------------------------|
| <i>Teacher</i> | <i>Course</i> | <i>con</i>               |
| $\omega_2$     | <i>Logic</i>  | $\omega_2 = Peter$       |
| <i>Paul</i>    | <i>Logic</i>  | $\neg(\omega_2 = Peter)$ |

Indeed, if  $\omega_2$  is interpreted by *Peter*, then relations in  $Rep(Teach')$  must contain the tuple  $\langle Peter, Logic \rangle$ , and if  $\omega_2$  is not interpreted by *Peter*, then corresponding relations in  $Rep(Teach')$  must contain the tuple  $\langle Paul, Logic \rangle$ . Thus, every relation in  $Rep(Teach')$  must contain either  $\langle Peter, Logic \rangle$  or  $\langle Paul, Logic \rangle$ . Of course, they may also contain both.

The first order theory associated with the table *Teach* is

$$DB_4 = \{Teach(John, Mathematics), \\ \exists x (\neg(x = Mathematics) \rightarrow Teach(Paul, x)), \\ \exists y ((y = Peter \vee y = Paul) \rightarrow Teach(y, Logic))\}$$

The Skolemized theory that corresponds to  $DB_4$  is

$$DB'_4 = \{Teach(John, Mathematics), \\ \neg(\omega_1 = Mathematics) \rightarrow Teach(Paul, \omega_1), \\ (\omega_2 = Peter \vee \omega_2 = Paul) \rightarrow Teach(\omega_2, Logic)\}$$

And the first order theory associated with *Teach'* is

$$DB'_4 = \{\exists x ((x = Peter \rightarrow Teach(x, Logic)) \wedge \\ (\neg(x = Peter) \rightarrow Teach(Paul, Logic)))\}$$

## *First Order Deductive Databases*

Relational databases have been extended to deductive databases containing rules, represented by first order Horn clauses [13]. Although this extension increased the expressive power of relational databases, it had no significant implications on the aspect of uncertainty arising from incompleteness.

A more significant change, from the point of view of uncertainty, was the extension to *Disjunctive Deductive Databases (DDDB)*, as defined by Minker *et al* [17, 19, 21].

A DDDDB is defined as a first order theory with two kinds of information: ground facts, that are positive ground clauses, and rules, that are non-Horn clauses. As usual in deductive databases, these clauses do not contain function symbols. The following is an example of a DDDDB.

$$\begin{aligned}
 DB_5 = & \{Teach(John, Mathematics), Teach(Peter, Logic), \\
 & Permanent(John), \\
 & \forall x \forall y (Professor(x) \vee Assistant(x) \leftarrow Teach(x, y) \wedge Permanent(x)) \\
 & \forall x (PhD(x) \leftarrow Professor(x)), \forall x (PhD(x) \leftarrow Assistant(x))\}
 \end{aligned}$$

## 2.2 Reasoning with Uncertain Information

Reasoning techniques are needed to compute answers to queries. The answer to a query represented by the first order formula  $q(x)$ , where  $x$  is the free variable in  $q$ , is defined as the set of constants in the language whose substitutions for  $x$  in  $q(x)$  gives a logical consequence of the theory  $DB$ . More formally, the answer is defined by

$$\{a \mid DB \vdash q(a)\}$$

This definition can easily be extended to queries that have more than one free variable.

In principle, any theorem proving technique may be used to compute answers. However, for efficiency, special techniques must be designed for the specific kind of theories encountered in intelligent databases. For this reason, researchers in relational databases proposed at first using the relational algebra to compute answers. Relational algebra (RA) is very efficient because it computes all the tuples that satisfy a given formula “at the same time”. Nevertheless, problems arise with RA.

The first problem is mainly theoretical: not all first order formulas can be translated to RA. For example, the query  $\neg Teach(x, Mathematics)$  cannot be translated. In fact, only *domain independent* formulas can be translated to RA (see [7]). Fortunately, queries that cannot be translated have little practical interest.

### *Relational Databases*

The second problem, a more serious one, is that in the context of the Open World Assumption, the relational algebra is not always sound. Consider, for

example, the query

$$Q_1(x) = \exists y (Teach(x, y) \wedge \neg Teach(x, Mathematics))$$

Its translation to RA is

$$Q_1 = \pi_{Teacher}(Teach - \sigma_{Course=Mathematics}(Teach))$$

Evaluating  $Q_1$  in the table  $Teach$  of Section 2.1.1 results in  $\{Peter\}$ , while we do not have

$$DB_1 \vdash \exists y (Teach(Peter, y) \wedge \neg Teach(Peter, Mathematics))$$

because  $\neg Teach(Peter, Mathematics)$  is not derivable from  $DB_1$ .

To get a sound and complete answer we should evaluate  $Q_1$  in every model of  $DB_1$  and retain in the answer only tuples that are in the intersection of all these evaluations. This, of course, is intractable in general.

In fact, a relational table can be considered a representative of all the models only for a certain kind of queries; intuitively, for the queries that without negation,<sup>2</sup> and only the logical connectives negation, disjunction and existential quantification. This corresponds, in RA, to queries that do not involve differences or negative selections. For these queries, evaluation on the tables indeed gives the same result as the intersection of all the evaluations in all the models.

## *Relational Databases with Null Values*

It was shown in [15] that RA operators can be extended to evaluate “correctly” certain kinds of queries on tables with null values.

To define a “correct” evaluation we must first define  $\Omega$ -equivalence between the representations of two tables  $T$  and  $U$ , where  $\Omega$  is a set of RA operators; for example, projection and union.  $T$  and  $U$  are  $\Omega$ -equivalent if, for RA expressions formed with operators in  $\Omega$ ,  $Rep(T)$  and  $Rep(U)$  are indiscernible, in the sense that the intersection of the evaluations of such expressions on relations in  $Rep(T)$  or in  $Rep(U)$  are identical. This equivalence relation is denoted by

$$Rep(T) \equiv_{\Omega} Rep(U)$$

---

<sup>2</sup>This characterization is somewhat informal, since a query like  $\neg \neg Teach(x, Mathematics)$  can, of course, be evaluated correctly on a table.

It is possible to “correctly” evaluate a RA expression  $exp$  on a table  $T$ , if it is possible to define a corresponding expression  $exp'$ , with specific extended RA operators for this type of table, in such a way that the representation of the evaluation of  $exp'$  on  $T$ ,  $Rep(exp'(T))$ , contains the same information, with respect to  $\Omega$ -operators, as the result of the evaluation of  $exp$  on all the relations in  $Rep(T)$ ; i.e.,<sup>3</sup>

$$Rep(exp'(T)) \equiv_{\Omega} exp(Rep(T))$$

Imielinski and Lipski showed that for tables with null values, queries formed with projection and selection can be “correctly” evaluated.

The projections in  $exp'$  are evaluated as usual. A selection of the form  $\sigma_{sel}(T)$  is evaluated as follows: a tuple  $t$  in  $T$  is in the result iff  $sel(t)$  is true for any substitution of all the nulls in  $sel(t)$  by any element of the domain. Notice that this definition does not require a 3-valued logic. In particular, if  $sel$  is a tautology such as  $(Teacher = Peter) \vee \neg(Teacher = Peter)$ , it will be evaluated to *true* even if, for a given tuple  $t$ , the value of  $Teacher$  is  $-$ . As an example, the query

$$q_2(x, y) = Teach(x, y) \wedge \neg(x = Peter)$$

whose translation to RA is

$$Q_2 = \sigma_{\neg(Teacher=Peter)}(Teach)$$

gives the following result when evaluated in the relation  $Teach$  of Section 2.1.2:

|             |                    |
|-------------|--------------------|
| <i>John</i> | <i>Mathematics</i> |
| <i>Paul</i> | $-$                |

It is important to notice that the result is also a table with null values. This means that in this formalism we can have incomplete information and incomplete answers as well. Indeed, the only standard answer to  $q_2$ ; i.e., the only tuple  $\langle a, b \rangle$  such that  $DB_2 \vdash q_2(a, b)$ , is  $\langle John, Mathematics \rangle$ . The second tuple in the table is an incomplete answer in the sense that

$$DB_2 \vdash \exists y (q_2(Paul, y))$$

### Relational databases with marked null values

The relational algebra was also extended to tables with marked null values [15]. The only extended operator whose definition differs from the standard is the

---

<sup>3</sup>To avoid excessive notation, we denote by  $exp(Rep(T))$  the *intersection* of the evaluation of  $exp$  on each relation in  $Rep(T)$ .

selection. To define this extension, a valuation  $v$  of the set of marked null values  $\omega_i$  is defined as an assignment of an element of the domain to each marked null value. Then, extended selection is defined by

$$\sigma_{sel}(T) = \{t \mid t \in T \wedge sel_*(t) = true\}$$

where  $sel_*(t) = true$ , if  $sel(v(t)) = true$  for every valuation  $v$ , and  $sel_*(t) = false$  otherwise.

It was shown in [15] that queries formed with projection, positive selection, union and Cartesian product, can be “correctly” evaluated. For example, the query

$$q_3(x) = Teach(x, Logic) \wedge Teach(x, Programming)$$

whose translation to RA is

$$Q_3 = \pi_1(\sigma_{2=Logic \wedge 4=Programming}(Teach \times Teach))$$

gives this result, when evaluated in the relation *Teach* of Section 2.1.3:

$$\boxed{\omega_2}$$

The standard answer is empty, because there is no constant  $a$  such that  $DB_3 \vdash q_3(a)$ . However, here again, the marked null value  $\omega_2$  in the result can be interpreted as an incomplete answer, since we have

$$DB_3 \vdash \exists x q_3(x)$$

Reiter showed in [28] that, in the context of marked null values, standard relational algebra is sound, but is not complete, in general, for the operators projections and union. Indeed, if we denote by  $\|q(x)\|$  the answer to  $q(x)$ , defined by

$$\|q(x)\| = \{a \mid DB \vdash q(a)\}$$

then we have

$$\begin{aligned} \pi_x \|P(x, y)\| &\subseteq \|\exists y P(x, y)\| \\ \|P(x, y)\| \cup \|Q(x, y)\| &\subseteq \|P(x, y) \vee Q(x, y)\| \end{aligned}$$

The equality of these expressions is not guaranteed.

### Relational databases with conditional tuples

The RA has also been extended in [15] to tables with conditional tuples. The extended projection is slightly different from the standard one. The difference comes from the fact that when tuples are projected on a set of attributes  $X$ , we have to keep trace of the conditions associated with these tuples:

$$\pi_X(T) = \{t[X \cup \{con\}] \mid t \in T\}$$

where  $t[X \cup \{con\}]$  denotes the projection of the tuple  $t$  on the attributes  $X \cup \{con\}$ .

The selection is also different from the standard one. The idea is to preserve all the tuples in the result of the selection, and to add, for each tuple, the selection condition to the previous condition. More formally, a selection is defined by

$$\sigma_{sel}(T) = \{\sigma_{sel}(t) \mid t \in T\}$$

where  $\sigma_{sel}(t)$  is defined by  $\sigma_{sel}(t)[X] = t[X]$ , where  $X$  is the set of attributes in  $T$  except  $con$ , and  $\sigma_{sel}(t)[con] = t[con] \wedge sel(t)$ .

The extended Cartesian product is modified in a similar way. For each tuple in the result the condition is the conjunction of the conditions in the operands:

$$T \times U = \{t[X - \{con\}] \times u[Y - \{con'\}] \times con \wedge con' \mid t \in T \wedge u \in U\}$$

where  $X$  (respectively,  $Y$ ) is the set of attributes in  $T$  (respectively,  $U$ ), and  $con$  (respectively,  $con'$ ) is the attribute of  $T$  (respectively,  $U$ ) that contains the conditions.

The union too must be slightly modified to remove duplicated tuples.

With these RA extensions it is possible to “correctly” evaluate queries formed with projection, selection, union and Cartesian product. For example, the query

$$q_4(x, y) = Teach(x, y) \wedge x = Paul$$

whose translation to RA is

$$Q_4 = \sigma_{Teacher=Paul}(Teach)$$

gives the following result when evaluated in the relation *Teach* of Section 2.1.4:

|            |             |  |
|------------|-------------|--|
| John       | Mathematics | $true \wedge (John = Paul)$  |
| Paul       | $\omega_1$  | $\neg(\omega_1 = Mathematics) \wedge (Paul = Paul)$                |
| $\omega_2$ | Logic       | $(\omega_2 = Peter) \vee \omega_2 = Paul \wedge (\omega_2 = Paul)$ |

The condition in the first line, due to the unique name axiom  $\neg(\text{John} = \text{Paul})$ , is equivalent to *false*. The condition in the second line is equivalent to  $\neg(\omega_1 = \text{Mathematics})$ , and the condition in the third line is equivalent to  $\omega_2 = \text{Paul}$ , because, due to the UNA,  $(\omega_2 = \text{Peter}) \wedge (\omega_2 = \text{Paul})$  is equivalent to *false*. Hence, a simplified representation of the result is

|             |              |                                       |
|-------------|--------------|---------------------------------------|
| <i>Paul</i> | $\omega_1$   | $\neg(\omega_1 = \text{Mathematics})$ |
| $\omega_2$  | <i>Logic</i> | $\omega_2 = \text{Paul}$              |

The meaning of the second line in the result is that when  $\omega_2$  is interpreted as *Paul*, then  $\text{Teach}(\text{Paul}, \text{Logic})$  is true. In other words, it is possible that Paul teaches Logic. The logical interpretation of the two lines in this incomplete answer is

$$\begin{aligned} DB_4 \vdash \exists y (\neg(y = \text{Mathematics}) \rightarrow q_4(\text{Paul}, y)) \\ DB_4 \vdash \exists x (x = \text{Paul} \rightarrow q_4(x, \text{Logic})) \end{aligned}$$

### *First Order Deductive Databases*

Techniques to compute answers efficiently have been defined for deductive databases on the basis of the definition of *Least Fixpoint (LFP)* operators. These operators compute all the ground clauses that can be inferred in one step from a set of rules and a set of ground clauses. The inference rule is a sort of hyperresolution that allows to infer a ground clause from a rule instance and a set of ground clauses.

Minker and Rajasekar have defined in [21] a LFP operator that computes sets of positive ground clauses. Consider the disjunctive deductive database  $DB_5$  of Section 2.1.5. In the first step, from

$$\begin{aligned} \text{Professor}(\text{John}) \vee \text{Assistant}(\text{John}) \leftarrow \\ \text{Teach}(\text{John}, \text{Mathematics}) \wedge \text{Permanent}(\text{John}) \\ \text{Teach}(\text{John}, \text{Mathematics}) \\ \text{Permanent}(\text{John}) \end{aligned}$$

we infer

$$\text{Professor}(\text{John}) \vee \text{Assistant}(\text{John})$$

In the second step, from

$$\begin{aligned} \text{PhD}(\text{John}) \leftarrow \text{Professor}(\text{John}) \\ \text{PhD}(\text{John}) \leftarrow \text{Assistant}(\text{John}) \\ \text{Professor}(\text{John}) \vee \text{Assistant}(\text{John}) \end{aligned}$$

we infer

$$\begin{aligned} & PhD(John) \vee Assistant(John) \\ & PhD(John) \vee Professor(John) \end{aligned}$$

And, in the third step, from

$$\begin{aligned} & PhD(John) \leftarrow Professor(John) \\ & PhD(John) \vee Professor(John) \end{aligned}$$

we infer

$$PhD(John)$$

Hence, this LFP operator can be used to compute, for example, the answer to the query

$$q_5(x) = PhD(x)$$

Indeed, the only answer is

$$DB_5 \vdash PhD(John)$$

and it corresponds to what is computed by the LFP operator.

This LFP operator was then improved to concentrate on the derivation of consequences that are relevant to the query. Demolombe [5] defined a LFP operator that simulates the combination of backward chaining and forward chaining techniques for DDDBs.

In this context of disjunctive deductive databases, incomplete answers were defined in [6], where they are called *conditional answers*. The conditional answer to a query  $q(x)$  is a set of ground formulas of the form  $q(a) \leftarrow cond(a, b)$ , where  $cond(a, b)$  represents the *additional* information necessary to derive the answer  $q(a)$ . A general definition of conditional answers is

$$\{q(a) \leftarrow cond(a, b) \mid DB \vdash q(a) \leftarrow cond(a, b)\}$$

To remove trivial conditional answers like  $q(a) \leftarrow q(a)$ , or inconsistent conditional answers where  $cond(a, b)$  is inconsistent with  $DB$ , two additional conditions are imposed on conditional answers:

1.  $q(a) \leftarrow cond(a, b)$  is not a tautology.
2. There is no  $DB$  consequence that strictly subsumes  $q(a) \leftarrow cond(a, b)$ .

For example,  $PhD(Peter) \leftarrow Permanent(Peter)$  is a conditional answer to the query  $q_5(x)$ . Indeed, we have

$$\begin{aligned} DB_5 &\vdash PhD(Peter) \leftarrow Permanent(Peter) \\ DB_5 &\not\vdash PhD(Peter) \\ DB_5 &\not\vdash \neg Permanent(Peter) \end{aligned}$$

If we present the query

$$q_6(x) = Professor(x)$$

to the database  $DB_5$ , we get the conditional answer

$$\begin{aligned} DB_5 &\vdash q_6(John) \leftarrow \neg Assistant(John) \\ DB_5 &\vdash q_6(Peter) \leftarrow Permanent(Peter) \wedge \neg Assistant(Peter) \end{aligned}$$

It is worth noting that in the derivation of conditional answers, negation has its standard meaning, and is not interpreted by default. No assumption is made during the derivation of answers. The idea of conditional answers is to allow the user to assume that, for example, John is not an assistant, or that Peter has a permanent position but is not an assistant. The role of the derivation technique is only to determine the weakest assumption the user has to make to guarantee that a given object is in the standard answer.

The technique for computing conditional answers is based on a specific inference rule described in [11]. The general idea is to represent the information in clausal form, and to infer consequences, with a variant of the resolution principle, that preserve  $q(x)$  or instances of  $q(x)$ . So, if we start from the clauses in the database that contain  $q(x)$  or instances of  $q(x)$ , of the form  $q(x) \vee c(x)$ , we resolve these clauses with literals in the part  $c(x)$ , in order to find the resolvents such that the part  $c(x)$  is minimal with regard to subsumption. These last clauses contain the least incomplete information we can derive about  $q(x)$  or instances of  $q(x)$ .

## 2.3 Sources of Incompleteness

One source of incompleteness is total lack of information. For example, we might have no information about the fact  $Teach(Paul, Logic)$ .

Another source of incompleteness is having only partial information. In most cases, this partial information is a consequence of observed atomic facts, and

has the form of existentially quantified variables, or of disjunctions. For example, we may know the fact  $Professor(John)$ , and the rule  $\forall x (Professor(x) \rightarrow \exists y Teach(x, y))$ , and thereby infer the partial information  $\exists y Teach(John, y)$ . This rule might not be represented in the database, and a user might add  $\exists y Teach(John, y)$  “manually” to the database.

Disjunctive partial information too might be a consequence of direct observations and rules. For example, from the observed facts  $Teach(Peter, Logic)$  and  $Permanent(Peter)$ , and the rule  $\forall x \forall y (Teach(x, y) \wedge Permanent(x) \rightarrow Professor(x) \vee Assistant(x))$ , we infer  $Professor(Peter) \vee Assistant(Peter)$ .

In the case of quantitative information, like the weight of a person, the measured value might be obtained from scales that are known to have accuracy within 1%. Hence, when the weight indicated is 73kg, we infer that the true value is between 72 and 74. As another example, a measured radar signal, after complex numerical treatment, may lead us to conclude that the aircraft is either F16 or Mirage2000.

Partial information may be richer than a simple disjunction. In the previous example, we may know that the possibility that the aircraft is F16 is 0.9, and the possibility that it is Mirage2000 is 0.85.

## 2.4 Uncertainty Constraints

When incompleteness is judged not to be acceptable for particular types of information, a formalism is needed to express constraints that exclude it from the database.

For example, if salaries are represented in the database  $DB_5$ , one may want to require that for each teacher in the database, a salary is included in the database as well. A rule of the form

$$(\exists z Salary(x, z) \leftarrow Teach(x, y)) \forall x \forall y$$

is not the correct expression of this constraint. This rule denotes that for each teacher a salary value exists (in the world), but it does not indicate that this value is included in  $DB$ . We need to express that for each  $x$  that  $DB$  knows to be a teacher,  $DB$  knows the corresponding salary  $z$ , as well.

Reiter [30] introduced an epistemic modality to express this very kind of constraint. This modality is called  $B$  because, in general, a database is a collection

of *beliefs*, not necessarily of *facts*. Using this modality, the constraint in the example becomes

$$(\exists z B(\text{Salary}(x, z)) \leftarrow B(\text{Teach}(x, y))) \forall x \forall y$$

This epistemic modality distinguishes between  $B(\exists x \text{Salary}(\text{John}, x))$  that asserts that it is believed (by the database) that John has (in the world) a salary, and  $\exists x B(\text{Salary}(\text{John}, x))$ , that asserts that there exists a value  $x$  that is believed (by the database) to be John's salary.

This modality can also be used to express the constraint that for each teacher known by the database to have a permanent position, the database knows whether he is a professor or an assistant:

$$(B(\text{Professor}(x)) \vee B(\text{Assistant}(x)) \leftarrow B(\text{Teacher}(x, y) \wedge \text{Permanent}(x))) \forall x$$

A constraint of this form is satisfied if in each model of  $DB$ , in the logic of this modality, the constraint is true.<sup>4</sup>

### 3 VALIDITY

#### 3.1 Representation of Uncertainty

The information in the database  $DB$  is not necessarily a correct description of the world. Yet, we may have meta-information that tells us which parts of  $DB$  are guaranteed to be valid descriptions.

##### *Using Meta-Relations to Store Validity Views*

A first method to represent this meta-information was designed by Motro [23].<sup>5</sup> In this method, the valid information is characterized in terms of relational database *view definitions*, that are first order formulas, and in terms of sets of tuples that satisfy certain conditions.

---

<sup>4</sup>The epistemic logic KFOPCE used by Reiter for  $B$ , and defined by Levesque, is not described here.

<sup>5</sup>This method was designed by Motro for representing meta-information about both completeness and validity. Due to space limitations, we only cover here its application to validity. We note that Motro now prefers the term *soundness* for validity; see Chapter 2.

These conditions are expressed with tuples in *meta-relations*, which mirror the relations of the database. For each database relation  $R$ , the corresponding meta-relation will be denoted  $M.R$ . The tuples in  $M.R$  may contain variables or constants, and, roughly speaking, all the tuples in  $R$  that are instances of tuples in  $M.R$  are guaranteed to be valid.

As an example, consider a database with relations  $Nationality(Citizen, Country)$ , and  $Born(Person, Place, Date)$ , and assume that the valid parts of this database are characterized in these four views:

1.  $V_1(x, x_1) = \exists v (Nationality(x, x_1) \wedge Born(x, x_1, v) \wedge x_1 = China)$
2.  $V_2(y, x_2) = Nationality(y, x_2) \wedge x_2 = France$
3.  $V_3(u, x_3) = Nationality(u, x_3) \wedge x_3 = USA$
4.  $V_4(z, x_4, t) = Born(z, x_4, t) \wedge x_4 = France$

$V_1$  expresses that for people born in China and whose nationality is also Chinese, the name and the nationality (and place of birth) are guaranteed to be valid. Note that, for the same individuals, the date of birth is not guaranteed to be valid.  $V_2$  and  $V_3$  express that the names and nationalities are valid for persons of French or American nationality.  $V_4$  expresses that for people born in France, the name, place of birth and date of birth are valid.

In general, views are conjunctive formulas, possibly with existential quantifiers, without negation except for selection predicates. Selection predicates are conjunctions of comparisons, such as equality, greater than, and so on. It is the values of the free variables (the “subject” of a views) that are guaranteed to be valid.

These view definitions are stored as *meta-tuples* in the meta-relations. First, we assure that the variables used in the entire set of views are named differently. Then, each atomic formula of every view definition (other than comparison predicates) is stored as a meta-relation tuple. In these tuples, free variables are suffixed by a \*. For example,  $x$ , in view  $V_1$ , is “starred”, while  $v$  is not. Intuitively, a starred variable indicates a valid value. Finally, atomic formulas formed with the equality predicate, i.e.,  $x = a$  or  $x = y$ , are “stored” by substituting  $x$  by  $a$  or by  $y$  throughout the meta-database. Atomic formulas formed with comparison predicates other than equality (e.g.,  $x > a$ ) are stored in an *ad hoc* meta-relation called *Comparison*. For simplicity, we assume only equality predicates, and therefore ignore this meta-relation.

The meta-database for this example is

| <i>M.Nationality</i> |                |
|----------------------|----------------|
| <i>Citizen</i>       | <i>Country</i> |
| <i>x*</i>            | <i>China*</i>  |
| <i>y*</i>            | <i>France*</i> |
| <i>u*</i>            | <i>USA*</i>    |

| <i>M.Born</i> |                |             |
|---------------|----------------|-------------|
| <i>Person</i> | <i>Place</i>   | <i>Date</i> |
| <i>x*</i>     | <i>China*</i>  | <i>v</i>    |
| <i>z*</i>     | <i>France*</i> | <i>t*</i>   |

A possible instance of this database is

| <i>Nationality</i> |                |
|--------------------|----------------|
| <i>Citizen</i>     | <i>Country</i> |
| <i>Lee</i>         | <i>China</i>   |
| <i>Yang</i>        | <i>China</i>   |
| <i>Pierre</i>      | <i>France</i>  |

| <i>Born</i>   |               |             |
|---------------|---------------|-------------|
| <i>Person</i> | <i>Place</i>  | <i>Date</i> |
| <i>Paul</i>   | <i>USA</i>    | <i>1955</i> |
| <i>Yang</i>   | <i>China</i>  | <i>1920</i> |
| <i>Pierre</i> | <i>France</i> | <i>1968</i> |

According to the meta-database, Pierre's date of birth is valid, whereas Yang's date of birth is not guaranteed to be valid. Yang's nationality is guaranteed to be valid, whereas Lee's nationality is not, because while the database includes a *Nationality* tuple for Lee it does not include a *Born* tuple for him, which is necessary to express the validity of his nationality. This latter example shows that to determine if a value is valid, it is not enough to consider the corresponding meta-relation; the entire meta-database may need to be considered.

### *Using Epistemic Logic to Define Reliable Information*

In another approach, presented by Demolombe and Jones [9], information validity depends on the type of information, and on the agent who stored the information in the database. The word agent should be understood in its broad sense: it may refer to a particular user, or a category of users, or, in the context of distributed databases, a particular database.

The formalization of information validity takes inspiration from signaling act theory. It is expressed in the framework of epistemic logic and logic of actions, and it covers databases that are sets of propositional calculus sentences.

A database is considered a communicative tool between agents that store information in it database, and agents that read information from it. The formalization presented in [9] allows complex situations where the representation is

not necessarily explicit sentences, but possibly any kind of encoded messages. The agents who interpret messages can also be represented explicitly. These agents are not necessarily reliable, but are assumed to be sincere.

The meta-information about validity defines the agents who are reliable when they assert; i.e., when they store sentences in the database. An informal example of such meta-information is “Agent Shu is reliable when he stores the fact  $Nationality(Yang, China)$ ”. The fact  $Nationality(Yang, China)$  is not necessarily present in the database, but, if present, is guaranteed to be valid. In a simplified version of the formalism presented in [9] this meta-information is expressed thusly

$$In.DB(Shu, Nationality(Yang, China)) \rightarrow Nationality(Yang, China)$$

meaning that if the message  $Nationality(Yang, China)$  is inserted by Shu, then  $Nationality(Yang, China)$  is valid. This formula is abbreviated

$$In.MDB(Shu, Nationality(Yang, China))$$

An extension of this formalism to first order logic (FOL) allows meta-information like

$$\forall x (In.DB(Shu, Nationality(x, China)) \rightarrow Nationality(x, China))$$

abbreviated in

$$In.MDB(Shu, Nationality(x, China))$$

And, in general,

$$In.MDB(agt, View(x)) \stackrel{def}{=} \forall x (In.DB(agt, View(x)) \rightarrow View(x))$$

where  $View(x)$  is any first order formula. For a particular  $View(a)$ , the meaning of  $In.DB(agt, View(a))$  is that  $View(a)$  was stored by agent  $agt$ ; i.e., all the facts of  $View(a)$  were inserted in one transaction activated by agent  $agt$ .

## 3.2 Reasoning with Uncertain Information

### *Deriving Valid Answers Algebraically*

Motro defined a method for reasoning about the views that define the valid information. This method extends the relational algebra in a relatively simple

and elegant way, to compute the meta-tuples that define the tuples in an answer that are guaranteed to be valid.

Assume a RA query (involving projection, selection and Cartesian product) is expressed by the RA expression  $exp$ , and let  $exp(DB)$  denote its answer. Denote  $exp'$  the corresponding expression in the extended RA. Let  $meta$  denote a mapping that assigns each relation  $R$  its meta-relation counterpart  $M.R$ . Then  $meta(exp(DB))$  denotes the meta-tuples that characterize the valid tuples in the answer  $exp(DB)$ . The latter meta-relation can be computed by evaluating  $exp'$  in the meta-database  $meta(DB)$ ; i.e.,  $exp'(meta(DB))$ . In formal terms,  $meta$  is a morphism for the above mentioned operators:

$$meta(exp(DB)) = exp'(meta(DB))$$

The three RA operators are extended as follows:<sup>6</sup>

1.  $M.R \times M.S = \{ \langle r, s \rangle \mid (r \in M.R) \wedge s \in M.S \}$
2.  $\pi_{X-A} = \{ t[X-A] \mid (t \in M.R) \wedge ((t[A] = v*) \vee (t[A] = v)) \wedge (v \text{ is unconstrained}) \}$   
where  $X$  is the set of attributes of  $M.R$ ,  $A$  is the attribute of  $M.R$  being removed,  $v$  is a variable symbol, and “ $v$  is unconstrained” means that  $v$  does not occur elsewhere in the meta-database.
3.  $\sigma_{A=a}(M.R) = \{ t.s \mid (t \in M.R) \wedge (t[A] \text{ is starred}) \wedge (t[A] \text{ is unifiable with } a) \}$   
where  $s$  is the unifier of  $t[A]$  and  $a$ .  
 $\sigma_{A=B}(M.R) = \{ t.s \mid (t \in M.R) \wedge (t[A] \text{ is starred}) \wedge (t[B] \text{ is starred}) \wedge (t[A] \text{ is unifiable with } t[B]) \}$   
where  $s$  is the unifier of  $t[A]$  and  $t[B]$ .

As an example, consider the RA query  $Q_7 = \sigma_{Place=China}(Born)$ . Its FOL representation is

$$q_7(x, y, z) = Born(x, y, z) \wedge (y = China)$$

When evaluated in the meta-database described earlier its meta-answer is

|       |           |     |
|-------|-----------|-----|
| $x^*$ | $China^*$ | $v$ |
|-------|-----------|-----|

<sup>6</sup>These definitions are slightly different of the definitions given by Motro, but they are compatible if we consider only the equality comparison predicate.

This single tuple meta-answer characterizes the tuples in the answer that are guaranteed to be valid. The meta-tuple indicates that the name and place of birth (in this case, China) are guaranteed to be valid; note that the date of birth is not guaranteed to be valid. It is worth noting that this characterization is independent of the contents of the database.

As another example, consider the RA query  $Q_8 = \sigma_{Date=1920}(Born)$ . Its FOL representation is

$$q_8(x, y, z) = Born(x, y, z) \wedge z = 1920$$

Its meta-answer is

|       |            |       |
|-------|------------|-------|
| $z^*$ | $France^*$ | $t^*$ |
|-------|------------|-------|

The tuple  $\langle x^*, China^*, v \rangle$  is not in the meta-answer, because date of birth is not guaranteed to be valid for people born in China. Thus,  $\langle Yang, China, 1920 \rangle$  is in the answer, yet without any guarantee of its validity.

As a third example, consider the RA query  $Q_9 = \pi_{Country}(Nationality)$ . Its FOL representation is

$$q_9(x, y) = \exists x Nationality(x, y)$$

Its meta-answer is

|            |
|------------|
| $France^*$ |
| $USA^*$    |

The tuple  $\langle China^* \rangle$  is not in the result, because the variable  $x$  is constrained by conditions in  $M.Born$ . If  $\langle China^* \rangle$  were in the result, it would imply that if the value  $China$  appears in an answer then this value is valid. We can see that this is not necessarily the case, because, for example, for Lee, the value of the attribute  $Country$  is not guaranteed to be valid.

Consider now the RA query  $Q_{10} = \pi_{Citizen, Country, Person, Place}(\sigma_{Citizen=Person}(\sigma_{Country=USA}(Nationality)) \times Born)$ . Its FOL representation is

$$q_{10}(x, y, z) = \exists u (Nationality(x, y) \wedge y = USA \wedge Born(z, t, u))$$

The result of  $\sigma_{Nation=USA}(Nationality)$  is

|       |         |
|-------|---------|
| $u^*$ | $USA^*$ |
|-------|---------|

The result of  $\sigma_{Country=USA}(Nationality) \times Born$  is:

|       |         |       |            |       |
|-------|---------|-------|------------|-------|
| $u^*$ | $USA^*$ | $x^*$ | $China^*$  | $v$   |
| $u^*$ | $USA^*$ | $z^*$ | $France^*$ | $t^*$ |

The result of  $Q_{10} = \sigma_{Citizen=Person}((\sigma_{Nation=USA}(Nationality)) \times Born)$  is

|       |         |       |            |       |
|-------|---------|-------|------------|-------|
| $u^*$ | $USA^*$ | $u^*$ | $China^*$  | $v$   |
| $u^*$ | $USA^*$ | $u^*$ | $France^*$ | $t^*$ |

And the final result is

|       |         |       |            |
|-------|---------|-------|------------|
| $u^*$ | $USA^*$ | $u^*$ | $China^*$  |
| $u^*$ | $USA^*$ | $u^*$ | $France^*$ |

Notice that if the projection:  $\pi_{Citizen,Country,Place}(M.R)$  is now applied to this final result, the result will be empty. Because both variables in the column *Person* are constrained, neither tuples survives the projection.

It is interesting to notice that the method designed by Motro cannot work for queries with the union operator. Indeed, *meta* is not a morphism for the union. Consider, for example, the RA query  $Q_{11} = Nationality \cup (\pi_{Person,Place}(Born))$ . Evaluating the union in the meta-database “as usual” yields

|       |            |
|-------|------------|
| $x^*$ | $China^*$  |
| $y^*$ | $France^*$ |
| $u^*$ | $USA^*$    |

Whereas the answer obtained from the database itself is

|               |               |
|---------------|---------------|
| <i>Lee</i>    | <i>China</i>  |
| <i>Yang</i>   | <i>China</i>  |
| <i>Pierre</i> | <i>France</i> |
| <i>Paul</i>   | <i>USA</i>    |

According to this answer, and the meta-tuple  $\langle u^*, USA^* \rangle$ , we should be *guaranteed* that Paul has American nationality or was born in the USA, which is not the case. The only information about Paul in the database is that he was born in the USA, but no tuple in the meta-relation *Born* guarantees that this information is indeed valid. Later we shall explain why this method does not work for the union.

### Deriving Safe Answers Logically

In the method defined by Demolombe and Jones the meta-information is represented by sentences of the form  $In.MDB(agt, f)$ , whose meaning is<sup>7</sup>

$$In.MDB(agt, f) \stackrel{def}{=} In.DB(agt, f) \rightarrow f$$

The information stored in the database, represented by sentences of the form  $In.DB(agt, f)$ , is considered a set of database beliefs. This interpretation is expressed formally in the axiom schema

$$In.DB(agt, f) \rightarrow B(f)$$

where the epistemic modality is assigned the standard semantics defined in the logic KD.

If we denote  $mdb$  and  $db$  the set of sentences of the form  $In.MDB(agt, f)$ , and  $In.DB(agt, f)$ , respectively, then the answer to a standard query  $q(x)$  is

$$\{a \mid db \vdash B(q(a))\}$$

whereas the answer to a safety query  $q(x)$  is

$$\{a \mid mdb \cup db \vdash q(a)\}$$

A *safety query* retrieves the values  $a$  in the standard answer that are guaranteed to be valid. Notice that the answer to the safety query is an explicit set of elements, not formulas (views), as in Motro's approach<sup>8</sup>

Consider, for example, the meta-database  $mdb$

$$\begin{aligned} & In.MDB(agt_1, (PhD(x) \leftarrow Professor(x)) \forall x) \\ & In.MDB(agt_2, Professor(x)) \end{aligned}$$

---

<sup>7</sup>In the definition given in [9], the sentence  $In.DB(agt, f) \rightarrow f$  is in the scope of a modal operator. This prevents paradoxes due to material implication.

<sup>8</sup>Though, by *evaluating* the views described in Motro's meta-answer in the standard answer, a similar result is obtained.

and the database  $db$

$$\begin{aligned} In.DB(agt_1, (PhD(x) \leftarrow Professor(x)) \forall x) \\ In.DB(agt_1, (PhD(x) \leftarrow Assistant(x)) \forall x) \\ In.DB(agt_2, Professor(John)) \\ In.DB(agt_2, Assistant(Peter)) \end{aligned}$$

From the  $mdb$  we can infer

$$\begin{aligned} (PhD(x) \leftarrow Professor(x)) \forall x \\ Professor(John) \end{aligned}$$

and from the  $db$  content we can infer

$$\begin{aligned} (PhD(x) \leftarrow Professor(x)) \forall x \\ B((PhD(x) \leftarrow Professor(x)) \forall x) \\ B((PhD(x) \leftarrow Assistant(x)) \forall x) \\ B(Professor(John)) \\ B(Assistant(Peter)) \end{aligned}$$

Hence, the answer to the standard query  $PhD(x)$ , represented by  $B(PhD(x))$ , is  $\{John, Peter\}$ , and the answer to the safety query  $PhD(x)$  is  $\{John\}$ .

This example shows that there may several possibilities to derive a fact of a predicate. Some derivations are based on reliable axioms and lead to valid conclusions, while other derivations involve non reliable axioms and only lead to beliefs. In the case of  $PhD(x)$ , it is not possible to guarantee that every fact of this form is valid. When the database concludes that someone has a PhD, because he is a professor, then this conclusion is valid, but it because he is an assistant, then this conclusion is not necessarily valid.

## *Comparison*

To compare the method developed by Motro with the method developed by Demolombe and Jones, we reformulate both in a common logical framework.

For simplicity, we ignore the agents who store the information, and we only distinguish between information that has been explicitly stored, which is considered a set of *explicit beliefs*, and the overall information, explicitly stored or derived, which is considered a set of *beliefs*. The meta-information characterizes beliefs that are *guaranteed* to be true in the world.

Consider first the method developed by Demolombe and Jones. Let  $EB(f)$  denote that  $f$  is an explicit belief. As we are ignoring the agent who stored  $f$

in the database (i.e.,  $f$  is valid independently of who stored it), we have the definition

$$In.MDB(f(x)) \stackrel{def}{=} \forall x (EB(f(x)) \rightarrow f(x))$$

where  $f(x)$  is any first order formula.

Moreover, we assume that the form of sentences that are explicitly stored is irrelevant. That is, if we have:  $f \leftrightarrow g$ , then we have  $EB(f) \leftrightarrow EB(g)$ .

The link between explicit and implicit beliefs is defined by the axiom schema:

$$EB(f) \rightarrow B(f)$$

Standard and safety queries are still represented by  $B(q(x))$  and  $q(x)$ , respectively.

In Motro's method, a validity view  $f(x)$  defines a part of the database that is guaranteed to be valid. Hence, for any  $x$ , if  $f(x)$  is either explicitly or implicitly believed by the database, then  $f(x)$  is true in the world. Denoting with  $M.DB(f(x))$  that the view  $f(x)$  is valid, we have

$$M.DB(f(x)) \stackrel{def}{=} \forall x (B(f(x)) \rightarrow f(x))$$

Of course, the link between explicit beliefs and implicit beliefs, is expressed by the same axiom schema.

Using these definitions we can easily rediscover the property that the mapping *meta* is a morphism for the extended RA operators projection, selection and Cartesian product:

In FOL projection is represented by existential quantification, and we have

$$M.DB(f(x, y)) \rightarrow M.DB(\exists y (f(x, y)))$$

In FOL selection is represented by Boolean expressions, denoted  $sel(x)$ , formed only with comparison predicates, and we have

$$M.DB(f(x, y)) \rightarrow M.DB(f(x, y) \wedge sel(x))$$

In FOL Cartesian product is represented by conjunctions, where the operands have no common free variable, and we have

$$M.DB(f(x, y)) \wedge M.DB(g(z, t)) \rightarrow M.DB(f(x, y) \wedge g(z, t))$$

However, we can easily verify that *meta* is not a morphism for the union, because the following property *does not* hold

$$M.DB(f(x, y)) \rightarrow M.DB(f(x, y) \vee g(x, y))$$

Intuitively, consider a tuple  $\langle a, b \rangle$ , and assume  $B(f(a, b) \vee g(a, b))$ , because  $B(g(a, b))$ . The meta-information  $M.DB(f(a, b))$ , whose meaning is  $B(f(a, b)) \rightarrow f(a, b)$ , does not allow to infer  $f(a, b) \vee g(a, b)$  when  $B(f(a, b))$  does not hold.

### 3.3 Sources of Validity Problems

The primary reason behind database information which is not valid are changes in the real world, that go unrecognized and therefore are not followed up by corresponding updates in the database. Thus, as time goes by, database information tends to lose its validity, which means that unless the information is updated, meta-information regarding validity must be retracted.

Another cause of invalid information is non-reliable sources of information. That is, when the agent who stores information in the database does not correctly transmit the messages he receives.

Another important cause of uncertainty is the systematic methods intended to reduce the incompleteness of the information stored in the database. These methods are called *Plausible Reasoning*. The idea is to define *a priori* the type of missing facts that can be assumed true. Such assumptions may be wrong and result in information whose validity is uncertain.

We briefly present here two kinds of plausible reasoning that are used in the context of intelligent databases: *default reasoning* [29, 3], and *inductive reasoning*.

#### *Default Reasoning*

An intuitive assumption accepted by many in the context of relational databases, is that there exists no true information which is not explicitly stored in the database (or is derivable from information stored in the database). This means that every sentence is either derivable from the database, or its *negation* is assumed, by default, to be derivable. This intuitive assumption was formalized by Reiter [27] with axioms called the *Closure World Assumption (CWA)*.

The CWA includes the UNA (see Section 2.1.1), *Domain Closure Axioms* that restrict the set of possible elements in a model to the set of elements that are actually present in the database, and *Completion Axioms* that restrict the set of possible true atomic facts in a model to the facts that are actually represented by tuples in the tables.

Consider, for example, the simple database presented in Section 2.1.1. The DCA is expressed as follows

$$\forall x ((x = John) \vee (x = Peter) \vee (x = Mathematics) \vee (x = Logic))$$

Such axioms allow to derive sentences with universal quantifiers. For example, with the DCA we can derive that every teacher teaches either Mathematics or Logic:

$$\forall x \forall y (Teach(x, y) \rightarrow Teach(x, Mathematics) \vee Teach(x, Logic))$$

In a sense, we have induced that a sentence is valid for every possible element, from the fact that it is true for the database elements.

For the same database, completion axioms express that the set of tuples that satisfy the predicate *Teach* is restricted to the tuples present in the table *Teach*:

$$\forall x \forall y (Teach(x, y) \rightarrow (x = John \wedge y = Mathematics) \vee (x = Peter \wedge y = Logic))$$

From this axiom, the UNA, and general properties of equality, we can formally infer

$$\neg Teach(Peter, Mathematics)$$

A relational database completed with the CWA has only one model, up to an isomorphism, or, in other terms, has only one Herbrand model, namely the minimal Herbrand model. In the same example, the database represented by the theory  $DB_1$  and CWA axioms, has the unique model  $D_1$ . Clearly, assuming the CWA is equivalent to selecting one of the possible models. Since this selection is based on formal criteria only, and ignores the semantics of the data, it is a source of errors. A more flexible assumption would be to assume completion axioms only for some of the predicates.

The CWA was extended by Reiter [27] to relational databases with marked null values. From a formal point of view the CWA is represented by the same type of axioms. The only change is that there are no unique name axioms for Skolem constants. For example, for the database represented by the theory  $DB_3$ , the

completion axiom of *Teacher* is

$$\begin{aligned} \forall x \forall y (Teach(x, y) \rightarrow & (x = John \wedge y = Mathematics) \vee \\ & (x = Paul \wedge y = \omega_1) \vee \\ & (x = \omega_2 \wedge y = Logic) \vee \\ & (x = \omega_2 \wedge y = Programming)) \end{aligned}$$

From the completed theory it is not possible to infer  $\neg Teach(Paul, Mathematics)$ , because we cannot infer  $\neg(\omega_1 = Logic)$ . In that case no choice is made about the course  $\omega_1$  taught by Paul.

The CWA was also extended by Reiter to disjunctive databases, that is, databases that contain ground positive clauses. Consider, for example, the disjunctive database

$$Professor(John), Professor(Peter) \vee Assistant(Peter)$$

This theory has models in which *Assistant(Peter)* is false, and then, at least in these models *Professor(Peter)* is true, and *Peter* is an element of the extension of *Professor*. The completion axiom for *Professor* is

$$\forall x (Professor(x) \rightarrow (x = John) \vee (x = Peter))$$

In general, the set of tuples that appear in completion axioms is the set of tuples that appear in some atom of some positive clause. The intuitive justification is that, if we have in the database a positive clause of the form  $P(a) \vee c$ , since the theory contains no negative literals, there exists a model in which  $c$  is false and  $P(a)$  is true.

Minker defined [20] the *Generalized Closed World Assumption (GCWA)*, whose proof theoretic definition is

$$\begin{aligned} DB \vdash_{GCWA} \neg A \text{ if for every positive clause } A \vee c : \\ DB \vdash A \vee c \text{ implies } DB \vdash c \end{aligned}$$

where  $c$  is a ground positive clause. This means that if there is no  $DB$  consequence of the form  $A \vee c$ , which is minimal with respect for subsumption, then we can infer  $\neg A$ .

The model theoretic definition of the GCWA is that an atom  $A$  in the Herbrand universe is false, if it is false in all the minimal Herbrand models of the database. More technically, if  $DB'$  is the database  $DB$  after the elimination of all subsumed clauses, and if  $atom(DB')$  represents all the atoms that appear

in a ground positive clause in  $DB'$ , then  $A$  is assumed to be false if it is not in  $atom(DB')$ .

This definition has been extended by Lobo, Rajasekar, and Minker [18] to disjunctive deductive databases. The definition is similar: we only have to replace  $DB'$  by the set of ground positive clauses in the LFP defined in [21] (see Section 2.2.5).

### *Inductive Reasoning*

Another source of uncertainty are techniques that extract rules from databases. The motivation is to have a synthetic representation of the information, which enables quick processing of certain queries, without scanning large sets of facts. The downside is that answers are not guaranteed to be valid. However, in most cases a quantitative estimation of the deviation is provided with the answer. Such rules and answers may be useful in applications like decision support systems. For example, in financial applications, large sets of data about stocks and bonds can be efficiently summarized with such rules. In many cases, such summaries are sufficient for decision making.

Induction techniques have been studied extensively in artificial intelligence, in particular machine learning. In the context of intelligent databases, they have been applied towards knowledge discovery and data mining [26] (see also Chapter 6).

A basic goal to *control* rule discovery. For example, a user defines the pattern of rules he is interested in, and a factor denoting the validity of the rule is computed on the basis of the database content. Another method for controlling the process of induction is to define an abstraction hierarchy on the constants or on the concepts [14, 12]. For example, in the relation *Born*, the details of the place of birth can be “abstracted” by the country of birth, and the details of the date of birth can be “abstracted” by the year of birth.

With these techniques, the fact  $Born(Yang, Canton, 02-09-1920)$  can be abstracted with  $Born(Yang, China, 02-09-1920)$ , then  $Born(Yang, China, 09-1920)$ , and then  $Born(Yang, China, 1920)$ . It is then possible to induce empirical rules such as

$$\forall x \forall y (Born(x, China, y) \rightarrow 1910 < y < 1930) [0.70, 0.85]$$

whose meaning is that among all the people born in China between 70% and 85% were born between 1910 and 1930. An inference method for logic programs with such rules in the case of monadic predicates was defined in [24].

Given the fact  $Born(Shu, Beijin, \omega_3)$ , the abstraction process and the inference process may be combined to infer that  $1910 < \omega_3 < 1930$ . Hence, inductive reasoning is reduce incompleteness.

### 3.4 Uncertainty Constraints

Initially, databases were used mainly in business applications, and validity was an important concern. This may explain the emphasis on integrity constraints in the relational model. Now the role of integrity constraints can be revisited in the context of intelligent databases. In [8] we considered the overall information in a database as a theory, and the issue was to define the role of integrity constraints in this general approach.

In our opinion, an important characteristic of the integrity constraints  $IC$  (within a database  $DB$ ) is that their validity should be guaranteed. Initially, relational integrity constraints were a function of the syntactical form of the sentences that represent them [25]. For example, non-Horn clauses were claimed to be, *a priori*, ICs, but this approach is clearly no longer valid. For example, in a disjunctive deductive database, non-Horn clauses can be used as well to derive information, or to check validity.

We consider  $IC$  a subset of the database  $DB$ . Although it is not certain that the world  $W$  is a model of  $DB$ , our assumption that the integrity constraints are always valid implies that  $W$  is a model of  $IC$ . Formally, this important property of  $IC$  is stated as follows

$$IC \subseteq DB \text{ and } W \in M(IC)$$

For this reason, the  $IC$  part of a database should not change when new information is acquired by the database, without any change in the world. If we know that the world has changed, it is assumed that  $IC$  is still valid. If some update  $UP$  leads to the new inconsistent database  $DB \cup UP$ , there are two possible attitudes.

It is possible to reject the update  $UP$ , because it may considered uncertain. This implies an implicit ordering on our confidence in the different kinds of

information: at the top level are integrity constraints, then the non-*IC* part of *DB*, and then the update *UP*. This may be denoted

$$UP < DB - IC < IC$$

Alternatively, we may enforce the update, and modify *DB - IC* to restore consistency. This attitude corresponds to the ordering

$$DB - IC < UP < IC$$

Consider this simple example:

$$\begin{aligned} DB = & \\ & \{\forall x (Male(x) \wedge Female(x) \rightarrow), \\ & \forall x (Professor(x) \rightarrow Male(x)), \\ & Professor(John)\} \\ IC = & \\ & \{\forall x (Male(x) \wedge Female(x) \rightarrow)\} \end{aligned}$$

and assume the update  $UP = \{Female(John)\}$ . Since  $DB \cup UP$  is inconsistent, we could either reject this update or modify it to restore consistency. Obviously, there are many possible ways to restore consistency; for example, we could remove either the fact  $Professor(John)$  or the rule  $(Male(x) \leftarrow Professor(x)) \forall x$ . Usually, one would prefer to restore consistency without removing rules.

The meta-information on information validity (Section 3.1) may be viewed as constraints about validity. For example, a sentence of the form

$$\forall x (EB(Professor(x)) \rightarrow Professor(x))$$

may be viewed as a description of a given state of the world and of the database content, and can be reformulated as “it is guaranteed that if the database believes that  $x$  is a professor, then  $x$  is indeed a professor”.

It can also be viewed as an obligation imposed to users who store information in the database with respect to the world, and it can be reformulated as “it should be guaranteed that whenever the database believes that  $x$  is a professor, then  $x$  is indeed a professor”. Nevertheless, it is important to notice that this constraint is different from the integrity constraints presented above. Indeed, integrity constraints *IC* are used to check the internal consistency of *DB*, whereas this meta-information, if it is considered as an obligation, can only be checked by a user, for example the database administrator. Consequently,

we propose calling the two kinds *internal integrity constraints* and *external integrity constraints*. These two kinds could coexist. For example, we may have in a single database

$$\begin{aligned}
 DB = & \\
 & \{\forall x (Male(x) \wedge Female(x) \rightarrow), \\
 & \forall x (EB(Professor(x)) \rightarrow Professor(x)), \\
 & EB(\forall x (Professor(x) \rightarrow Male(x))), \\
 & EB(Professor(John))\}
 \end{aligned}$$

where the first sentence is an internal integrity constraint, the second is an external integrity constraint, and the third and fourth are explicit beliefs. When processing the update  $Female(John)$ , since  $Professor(John)$  is guaranteed to be valid, this fact should not be removed to restore consistency.

## 4 CONCLUSION

In this chapter we surveyed two kinds of uncertainty: incompleteness and validity.

For incompleteness, the representations covered were traditional relational databases from the point of view of OWA, null values that represent existentially quantified variables, conditional tuples, and positive disjunctive facts. We described reasoning techniques that were based on relational algebra extensions for null values or conditional tuples, or on specific inference techniques supported by specific least fixpoint operators. We noted that the sources of uncertainty are either lack of information, or availability of partial information. Constraints to limit the incompleteness can be defined in terms of epistemic logic.

For validity, the representations covered were meta-information in the form of meta-tuples in meta-relations, or within a logical framework based on epistemic logic and logic of actions. The reasoning techniques were based on extensions to relational algebra for meta-relations, or on general inference techniques for non-classical logics. We noted that the origins of uncertainty include changes in the real world, as well as non-classical inference that involve plausible reasoning (default reasoning or inductive reasoning). Constraints to express limited validity are sentences that play a specific role with regard to database updates.

The overall attitude in the field of intelligent databases is to formulate and attack restricted problems, rather than broad, general problems (as is the ten-

dency in the field of artificial intelligence). This leads to solutions that are usually more efficient. For example, extensions of the relational algebra are often preferred over general inference methods. A drawback of this attitude is that it is hard to form a global view of the different techniques, and individual implementations might be difficult to integrate.

There is still much work to be done in the area of constraints. The distinction between descriptive sentences and normative sentences is not always clear for many researchers. That could be helped by using deontic logic [2].

In this chapter we considered only *qualitative* representation of uncertainty; a similar framework could be used to analyse the *quantitative* aspects of uncertainty. Similarly, for the other two kinds of uncertainty that we mentioned in the introduction: inconsistency and vagueness.

## REFERENCES

- [1] J. Biskup. A foundation of Codd's relational maybe-operations. *ACM Transactions on Database Systems*, 8(4), 1979.
- [2] J. Carmo and A. J. I. Jones. Deontic database constraints and the characterisation of recovery. In A. J. I. Jones and M. Sergot, editors, *2nd International Workshop on Deontic Logic in Computer Science*, pages 56–85. Tano A. S., 1994.
- [3] M. R. B. Clarke, C. Froidevaux, E. Grégoire, and P. Smets. Uncertainty conditionals and non-monotonicity. *Journal of Applied Non-Classical Logics*, 1(2), March 1991.
- [4] E. F. Codd. Extending the data base relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4), 1979.
- [5] R. Demolombe. An efficient evaluation strategy for non-Horn deductive data bases. In *IFIP Congress*. Elsevier, 1989. Extended version appeared in *Journal of Theoretical Computer Science*, No. 78.
- [6] R. Demolombe. A strategy for the computation of conditional answers. In *10th European Conference on Artificial Intelligence*, 1992.
- [7] R. Demolombe. Syntactical characterization of a subset of domain independent formulas. *Journal of the ACM*, 39(1), 1992.

- [8] R. Demolombe and A. Jones. Integrity constraints revisited. In A. Olive, editor, *4th International Workshop on the Deductive Approach to Information Systems and Databases*. Universitat Politecnica de Barcelona, 1993.
- [9] R. Demolombe and A. Jones. Deriving answers to safety queries. In R. Demolombe and T. Imielinski, editors, *Nonstandard queries and nonstandard answers*. Oxford University Press, 1994.
- [10] R. Demolombe and L. Fari nas del Cerro. An algebraic evaluation method for deduction in incomplete data bases. *Journal of Logic Programming*, 5:183–205, 1988.
- [11] R. Demolombe and L. Fari nas del Cerro. An inference rule for hypothesis generation. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1991.
- [12] V. Dhar and A. Tuzhilin. Abstract-driven pattern discovery in databses. 1992. Submitted for publication.
- [13] H. Gallaire and J. Minker. *Logic and Data Bases*. Plenum Press, 1978.
- [14] J. Han, Y. Cai, and N. Cercone. Knowledge discovery in databases: an attribute-oriented approach. 1992. Submitted for publication.
- [15] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4), 1984.
- [16] K. C. Liu and R. Sunderaman. Indefinite and maybe information in relational databases. *ACM Transactions on Database Systems*, 15(1), 1990.
- [17] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, 1992.
- [18] J. Lobo, A. Rajasekar, and J. Minker. Weak completion theory for non-Horn programmms. In R. A. Kowalski and K. A. Bowen, editors, *Proceedings of 5th International Conference on Logic Programming*. MIT Press, 1988.
- [19] J. Minker. Overview of disjunctive logic programming. *Annals of Mathematics and Artificial Intelligence*. To appear.
- [20] J. Minker. On indefinite databases and the closed world assumption. In *Lecture Notes in Computer Science, No. 138*. Springer-Verlag, 1982.
- [21] J. Minker and A. Rajasekar. Procedural interpretation of non-Horn logic programs. In *Proceedings of the Conference on Automated Deduction*, 1988.

- [22] A. Motro. Completeness information and its application to query processing. In *Proceedings of 12th International Conference on Very Large Data Bases*, 1986.
- [23] A. Motro. Integrity = validity + completeness. *ACM Transactions on Database Systems*, 14(4), 1989.
- [24] R. Ng and V. S. Subrahmanian. Empirical probabilities in monadic deductive databases. 1992. Submitted for publication.
- [25] J.-M. Nicolas and K. Yazdanian. Integrity checking in deductive databases. In H. Gallaire and J. Minker, editors, *Logic and Databases*. Plenum, 1982.
- [26] G. Piatetsky-Shapiro and W. J. Frawley. *Knowledge Discovery in Databases*. MIT Press, 1991.
- [27] R. Reiter. Towards a logical reconstruction of relational database theory. In *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*. Springer-Verlag, 1983.
- [28] R. Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *Journal of the ACM*, 33(2), 1986.
- [29] R. Reiter. Nonmonotonic reasoning. In *Annual Reviews of Computer Science*, 2, 1987.
- [30] R. Reiter. What should a database know? *Journal of Logic Programming*, 14(2,3), 1992.
- [31] Y. Vassiliou. Null values in data base management, a denotational semantics approach. In *Proceedings of ACM SIGMOD International Conference*, 1979.
- [32] K. B. Yue. A more general model for handling missing information in relational data bases using a 3-valued logic. *SIGMOD Record*, 20(3), 1991.