Logics for Databases and Information Systems

Edited by
Jan Chomicki, Gunter Saake
Monmouth University / University of Magdeburg

Kluwer Academic Publishers
Boston/Dordrecht/London

10 LOGICAL APPROACHES TO INCOMPLETE INFORMATION: A SURVEY

Ron van der Meyden

Abstract: The observation that it is frequently impossible to obtain complete information in the context of database applications has motivated a substantial literature seeking to extend the relational model. The paper surveys the literature on the modelling and processing of *incomplete* information (also known as *indefinite* information) using tools derived from classical logic and modal logic. The focus of the paper is on models in which query processing is decidable. Theories dealing with the quantification of indefiniteness, such as probabilistic or fuzzy models of uncertainty, are not discussed.

10.1 INTRODUCTION

It was noted early in the development of the database field that the data that need to be stored in a database frequently amount to an incomplete representation of the state of the application domain. For example, unknown values for some attributes of a data object were identified as being a common issue. This particular problem can arguably be handled by *ad hoc* means in most data models. However, it would be more elegantly addressed by a data model with a sufficiently rich semantics to allow the database system to support the maintenance of missing values by providing explicit constructs for their representation and manipulation. Indeed, missing values are just one of a number of issues motivating the exploration of more expressive data models [Cod79].

The relational data model [Cod70] has helped to provide a firm mathematical foundation for databases, based on the set theoretic construct of *relations*, and a corresponding set of operations called the *relational algebra*. It was natural to attempt to take this model as a basis for extensions with richer semantics. Early attempts to deal with incomplete information took the approach of extending the notions of relation and relational algebra to accommodate such missing values. These early proposals were soon seen to have difficulties, however, and it took some time before a fully adequate semantic model of missing values was developed.

This model was based on a slightly different view of the nature of a database, which takes it as representing a state of information about the world, rather than a direct model of the world. It is natural to express this state of information as a logical theory. In retrospect, it is not surprising that a solution to the problem of modelling data incompleteness should be based on a logical view of the nature of a database. The relational model of data was, via its basis in set theory, from its naisance seen to be closely related to first order logic [Cod72].

The modelling of missing values using tools from classical logic is now relatively well understood, and a substantial literature has explored the resulting models and their generalizations. In this paper, we survey this literature. The logical solution to the problem of missing values readily suggests a variety of more expressive data models, and the field can be viewed as blending naturally with the allied areas of logic programming, deductive databases and knowledge representation. To keep the paper to a manageable scope, we confine ourselves to works directly motivated by the desire to extend database technology to allow for the representation of incomplete information. Even within this scope it is difficult to give an exhaustive treatment, but we have endeavoured to provide a representative sample of citations from which the remainder of the literature can be accessed.

For the purpose of this paper we interpret "database technology" very broadly, to include some of the highly expressive types of data that have been studied in the literature on incomplete information. We take the distinguishing test between the database field and its generalizations to be the complexity of query processing – in a "database," query processing should be *decidable*. This is clearly an overly generous interpretation from the point of view of practical database systems, but it is a natural boundary from a theoretical point of view, particularly as the querying of incomplete information turns out to be inherently complex.

Our characterization excludes much of the areas of logic programming and knowledge representation which, through the use of function symbols or unrestricted syntax, do not guarantee the decidability of even simple queries. We also exclude from the scope of this paper types of data incompleteness that

cannot be represented by "classical" logical means. Thus, we do not consider models based on probability theory, rough sets, fuzzy logic, possibilities and other approaches that seek to quantify or rank the degree of uncertainty. The application of these areas to information systems is well treated in other surveys, e.g., [MS96]. We do briefly touch upon the use of modal and many-valued logics.

We assume that the reader is familiar with first order logic [Men64]. Section 10.2 discusses the motivation for the literature, describing a number of the sources of incompleteness of information in applications. Section 10.3 reviews the relational model of data and describes how it can be generalized to a semantic framework for incomplete information. The next two sections describe distinct ways in which the relational model can be generalized: Section 10.4 discusses approaches that seek to extend the relational algebra and Section 10.5 deals with logical databases, a more general approach based around the use of declarative query languages and representations of data. Section 10.6 discusses the complexity of query processing in both types of incomplete database. Negative information, which requires special treatment in incomplete databases, is the topic of Section 10.7. This is followed by two sections discussing the generalization of two areas of relational database theory to the context of incomplete databases: Section 10.8 deals with integrity constraints and Section 10.9 discusses updates. Section 10.10 considers a number of other topics arising in incomplete databases: inapplicable attributes, constraints, object orientation, schema design, approximation, and approaches based on modal and nonstandard logics. We conclude in Section 10.11 by describing the way incomplete information is currently handled in commercial relational database systems.

10.2 SOURCES OF INDEFINITENESS

The most common form of data incompleteness is missing information. Parties responsible for providing information frequently omit to give values for some attributes of data objects. The reasons are diverse, but may include carelessness, lack of knowledge or withholding of information due to privacy concerns. This leads to situations in which, e.g., a record relating to a Student "J. Ngu" needs to be stored in the database, but a value of the "Age" attribute for this record is unavailable.

A straightforward approach to such data incompleteness is to place the record in the database, but to place in the "Age" field a *null value*, a placeholder for the unknown value. This type of incompleteness was the earliest type identified [ANS75] and has been the object of the most extensive study in the literature. We treat this topic in Section 10.4. A closely related motivation for placing null values in a database arises in the *Universal Relation Model*, which

has been proposed as a means for the support of natural language interfaces to a database, and in the related *Weak Instance Approach*. This topic is briefly discussed in Section 10.10.1.

An indirect way in which incompleteness arises in a database is through *view updates* [Imi89]. Consider a database consisting of a relation *Employee* with attributes including *Name* and *Salary*. Not all users of the database will have access to the complete relation. For example, a junior staff member of the Human Resources Unit may be authorized to interact with the database only through a view of the relation that omits the *Salary* attribute of employee records. Suppose that updates are allowed on this view, and the user inserts a tuple for a new employee "Saliba". This update needs to be mapped back to a change to the underlying relation in such a way as to generate the inserted tuple. That is, a tuple should be inserted in the *Employee* relation with attribute *Name* equal to "Saliba". The problem is that this tuple also needs a value for the *Salary* attribute. We find ourselves here in a situation of incomplete information, since all possible values for this attribute would produce the desired view. A variety of approaches to this sort of problem have been proposed in the literature on view updates, but one approach that suggests itself in this case is to place a null value in this attribute. More complicated types of incomplete database are required once one moves beyond views defined using projections alone.

Another form of incompleteness concerns the relationship between the database and the external world it models. In some applications, the database could be viewed as *being* the world being modelled. For example, a university might decree that no person is a student for administrative purposes unless recorded in its database. Typically however, a database is intended to serve as a representation of an external reality. Even if all the facts recorded in the database are true, there may be additional facts that are not recorded. In some ways this is similar to missing information as described above, but the missing data here is at the level of *facts* rather than *values*. Various approaches to the representation of the extent of this type of incompleteness have been considered in the literature, an area we discuss in Section 10.7.

In recent years, the types of applications to which database systems are being applied have been extended beyond, e.g., commercial account-keeping applications to incorporate domains such as geographic information and planning and design data. Such domains have been a specific motivation of research aiming to develop spatial, object-oriented and temporal databases. Many of the sources of data incompleteness mentioned above apply in these extensions. However, the new applications require the support of much richer semantics, and accordingly generate new forms of indefinite information.

For example, in a scheduling system being used to maintain the plans for the construction of a building, it may have been determined that event F, laying the foundations, is to precede event W1, raising the Eastern Wall, and event W2, raising the Western wall, but no order may have been specified on W1 and W2. If we wish to answer a query about the sequence of events as it will actually occur, we are faced with a number of possibilities: the data is consistent with the two orders F,W1,W2 and F,W2,W1, and perhaps W1 and W2 may occur concurrently. In this scenario, even though all the remaining data may be complete, indefiniteness arises from incomplete information about the relative order of points in a linearly ordered domain. Indefinite database models expressing information of this sort are discussed in section 10.10.2.

A related source of indefiniteness that occurs in design applications concerns the representation of design choices [INV91a; INV91b]. Designs may be phrased in terms of abstract objects, for which a number of standard implementations are known to exist. For example, in a computer hardware design, a "multiplier" may be implemented in a variety of standard ways by combining "adders". In a top-down design methodology, one often delays decisions about the choice of implementation until the interactions of various choices are understood. The representation and querying of the intermediate stages of such a design process would be facilitated by a database system capable of representing the alternative implementations resulting from the available choices. An interesting generalization arises in this context when the lower level objects themselves are abstract, and may be implemented in a variety of ways. For example, "adders" may be implemented in different ways by combining NAND gates. This leads to a type of indefinite database in which the indefiniteness may be recursively unfolded [Lib94; Mey93].

10.3 A SEMANTIC FRAMEWORK FOR INCOMPLETE DATABASES

The considerations of the previous section suggest that database semantics should be sufficiently rich to provide the means to model incomplete information of the types mentioned. Of the proposals to this end that are based on logic, most can be understood as instances of a common semantic framework, which we present in this section. We begin by briefly recalling the relational model of data (see [Kan90] for a more comprehensive introduction), and describe how this model may be understood from a logical perspective. This leads us to a discussion of how this perspective may be generalized to yield a semantic framework for incomplete databases. We also discuss several distinct notions of query answer arising in the more general context.

Part	Supplier
$P33$	BigCo
$P34$	LittleCo
$P35$	LittleCo

Supplier	City
BigCo	New York
LittleCo	Middletown
Acme	Tuczon

Figure 10.1 Tables representing a relational database

10.3.1 The Relational Model

The relational model separates the *structure* of a database from its *contents.* Database structure is modelled using the idea of a *database scheme.* Suppose that At is a finite set of *attributes.* A database scheme $\mathcal{S}$ over At consists of a finite set $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ of *relation schemes*, where each relation scheme $\mathcal{R}_i$ is a subset of At. Informally, a database scheme provides a set of relation names, and describes the attributes applicable to each. Additionally, a database scheme may also contain a set of integrity constraints, but we defer discussion of this topic to Section 10.8.

To model database contents, the relational model uses the notion of *database instance.* Fix some set Δ, called the *domain*, to represent the possible data values. First, we define a *tuple* over a relation scheme $\mathcal{R}$ to be a mapping from $\mathcal{R}$ to values in Δ. (For simplicity we assume here that all attributes range over the same domain, but this is not essential.) An *instance of a relation scheme* $\mathcal{R}$ is then defined to be a *finite* set R of tuples over $\mathcal{R}$. We will refer to a relation instance simply as a *relation.* Finally, an *instance of a database scheme* $\mathcal{S} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ is defined to be a set $D = \{R_1, \ldots, R_n\}$ where each R_i is an instance of the corresponding relation scheme $\mathcal{R}_i$. It is common to depict a relational database instance as a set of tables, with the columns headed by attribute names. For example, Figure 10.1 illustrates an instance over the schema {{**Part**, **Supplier**}, {**Supplier**, **City**}}.

The relational model is associated with two types of query languages, a *procedural* language called the *relational algebra* and a *declarative* language, which is often called the *relational calculus*, but which amounts essentially to first order logic. The expressions of the relational algebra are built from a basic set of *relational operators*, defined as follows. Suppose that R_1 and R_2 are relations over relation schemes $\mathcal{R}_1$ and $\mathcal{R}_2$ respectively. Then

J. The *join* $R_1 \bowtie R_2$ is defined to be the relation over scheme $\mathcal{R}_1 \cup \mathcal{R}_2$ containing the tuple t just when there exist tuples $t_1 \in R_1$ and $t_2 \in R_2$

such that for each attribute $A \in \mathcal{R}_1$ we have $t(A) = t_1(A)$ and for each attribute $A \in \mathcal{R}_2$ we have $t(A) = t_2(A)$.

P. If X is a subset of $\mathcal{R}_1$ then the *projection* $\pi_X(R_1)$ is defined to be relation over scheme X containing the tuple t just when there exists a tuple $t_1 \in R_1$ such that for each attribute $A \in X$ we have $t(A) = t_1(A)$.

U. The *union* $R_1 \cup R_2$ is defined when $\mathcal{R}_1 = \mathcal{R}_2$, and is the relation over scheme $\mathcal{R}_1$ containing just those tuples that are either in R_1 or in R_2.

D. The *difference* $R_1 \setminus R_2$ is defined when $\mathcal{R}_1 = \mathcal{R}_2$, and is the relation over scheme $\mathcal{R}_1$ containing just those tuples that are in R_1 but not in R_2.

S. A *selection condition* E over the attributes of R_1 is a Boolean expression formed using the operators $\wedge, \vee$ and $\neg$ over atoms of the form $A = a$ or $A = B$, where a is a value in Δ and A and B are attributes of R_1. A selection condition is *positive* if it does not make use of the negation symbol $\neg$. If E is a selection condition over R_1 the *selection* $\sigma_E(R_1)$ is the relation over scheme $\mathcal{R}_1$ consisting of the tuples $t \in R_1$ such that the result of substituting $t(A)$ for each occurrence of an attribute A in E yields a boolean formula over equations of values in Δ that evaluates to **true** under the usual semantics.

R. If A and B are attributes then the *renaming* $\rho_{B|A}(R_1)$ is the relation over scheme $(\mathcal{R}_1 \setminus \{A\}) \cup \{B\}$ containing for each tuple t of R_1 the tuple t' defined by $t'(B) = t(A)$ and $t'(C) = t(C)$ for all other attributes C.

A *relational algebra expression* over a database schema $\mathcal{S}$ consists of an expression built up using these operators from a basis consisting of the relation schemas in $\mathcal{S}$. We will use combinations of the letters J, P, U, S, D and R from this list to refer to fragments of the relational algebra. Thus, "SPJ-query" refers to a relational algebra expression formed using the operations of selection, projection, and join, but not making use of union, difference or renaming. We use S^+ to refer to selections with positive conditions.

A relational algebra expression Q defines a function from database instances to relations in an obvious way. Given a database D, we simply substitute for each occurrence of the relation scheme $\mathcal{R}_i$ the corresponding relations R_i of D, and compute the result according to the definitions above. This completes the definition of the relational algebra.

Next, we define the *relational calculus,* which amounts essentially to first order logic. Note first that a database instance $D = \{R_1, \ldots, R_n\}$ can be interpreted in an obvious fashion as a first order structure (or *model*) $M_D = \langle U, R_1, \ldots, R_n \rangle$ for a language containing an k-ary relation symbol for each

relation of D with k attributes. We write R_i ambiguously to denote either the relation or the corresponding relation symbol — the meaning will always be clear from the context. The relations in M_D are just the relations in D (with some ordering imposed on the attributes). The one issue we need to address is what to take as the universe U of this structure. We will take this to be the set of values occurring in some R_i. (Other choices are possible, however, and this involves some subtleties concerning whether databases are finite, as we have assumed, or may be infinite: see [Kan90] for a discussion of these issues.)

Let $\Phi(\mathbf{x})$ be a formula of first order logic with free variables $\mathbf{x}$. Then a *relational calculus expression* is an expression Q of the form $\{\mathbf{x} : \Phi(\mathbf{x})\}$. Such an expression defines a function from databases instances D to k-ary relations, where k is the number of variables in $\mathbf{x}$, by

$$Q(D) = \{\mathbf{a} \mid M_D \models \Phi(\mathbf{a})\}$$

where '$\models$' denotes the usual notion of satisfaction from first order logic. That is, the answer of Q on database D is the set of tuples of values $\mathbf{a}$ such that the closed formula $\Phi(\mathbf{a})$ is satisfied in M_D. For brevity, we will write simply $\Phi(\mathbf{x})$ for $\{\mathbf{x} : \Phi(\mathbf{x})\}$, assuming that $\mathbf{x}$ lists all the free variables in the formula.

A fundamental result of database theory states that the relational algebra and the *safe* fragment of the relational calculus (a class of queries whose answers are independent of the choice of domain Δ) have equivalent expressive power [Cod72]. That is, for each relational algebra expression Q there exists a safe relational calculus expression Q' such that $Q(D) = Q'(D)$ for all databases D, and vice versa. Moreover, various fragments of the relational algebra correspond in expressive power to subclasses of first order logic. A *conjunctive* formula is a formula of the form $\exists x_1 \ldots \exists x_n[A_1 \wedge \ldots \wedge A_m]$, where the A_i are atomic formulae. A *positive existential* formula is a disjunction $C_1 \vee \ldots \vee C_m$, where the C_i are conjunctive formulae. Conjunctive formulae correspond to the class of S^bPJR-queries (where S^b refers to the set of *basic* selections in which conditions are of the form $A = a$ or $A = B$ only), and positive existential formulae to the class of S^+PJRU-queries. These equivalence results provide a way of understanding relational databases in terms of first order logic: a database consists of a model, and a query consists of a formula. This perspective is known as the *model theoretic* view of databases.

10.3.2 Incomplete Database Semantics

If a relational database, representing *complete information*, semantically corresponds to a first order structure, what is the semantics of *incomplete* information? Many approaches in the literature can be understood as answering this question by taking the semantic object corresponding to an incomplete data-

base to be a *set* $\mathcal{M}$ of (first order) structures. That is, an incomplete database corresponds to a *state of information*. Intuitively, the information encoded by $\mathcal{M}$ is that "the actual state of the world is one of the states in $\mathcal{M}$, but it is not known which." We may call this perspective the *information theoretic* or the *possible worlds model*.

We will take the information theoretic model as the basis for our characterizations of incomplete databases in this paper. Most treatments of incomplete information that are satisfactory from a logical perspective can be described using this view. Thus, at the most general level, we can say that a theory of incomplete databases specifies

1. a set $\mathcal{D}$ of admissible database instances,
2. a set $\mathcal{Q}$ of admissible queries,
3. a set $\mathcal{A}$ of admissible query answers,
4. a function *Mod* mapping each database instance in $\mathcal{D}$ to a *set* of first order structures, and
5. a procedure $Ans(D, Q)$ taking as input a database $D \in \mathcal{D}$ and a query $Q \in \mathcal{Q}$ and returning an answer in $\mathcal{A}$.

The answer $Ans(D, Q)$ should satisfy some correctness condition with respect to the semantics provided by *Mod*. The precise form of this condition will depend on the nature of the answer set. The next section discusses a number of the alternatives that have been considered.

The literature has focused on two main approaches to the definition of these attributes of a theory of incomplete databases. The first, discussed in Section 10.4, views database instances as formed from generalized types of tables. The second, which views database instances as logical theories, is discussed in Section 10.5.

It is worth commenting at this point on terminology. In the literature, one finds both the expressions "incomplete database" and "indefinite database" in use. It is useful to draw a distinction between these, in such a way as to preserve consistency with the meaning of the term "definite" within the logic programming literature. It is often possible to place a partial order $\leq$ on the class of structures of interest that captures a notion of "an increasing number of facts hold" in the sense that for positive existential queries Q, if $D \leq D'$ then $Q(D) \subseteq Q(D')$. For example, if one restricts attention to Herbrand models the set inclusion ordering has this property. (A Herbrand model is a first order structure with universe equal to the set of terms of the language without variables, with each such term interpreted as itself. Such models can

be represented as sets of facts without variables. See Chapter 2 for a precise definition.) Given such an order, we say that a database D

1. is *complete* if $Mod(D)$ contains a single model (or a single model up to isomorphism), and
2. is *definite* if $Mod(D)$ contains a unique model minimal under the order $\leq$, i.e., a model M such that $M \leq M'$ for all $M' \in Mod(D)$.

For example, if D is a database with $Mod(D)$ equal to all (Herbrand) models of the theory $\{A(a), \forall x(B(x) \Rightarrow A(x))\}$ then D is incomplete but definite. If D is a database with $Mod(D)$ equal to all (Herbrand) models of the theory $\{P(a) \vee P(b)\}$ then D is both incomplete and indefinite. The distinction is useful because definiteness often implies that query processing is tractable, whereas indefiniteness is very frequently the source of intractability.

10.3.3 Notions of Query Answer

In the relational model the result of applying a query to a database is a relation, i.e., a set of tuples. A tuple is an *answer* of the query if it is contained in this set. In generalizing the relational model, it is necessary to generalize the notion of answer. A number of different approaches are to be found in the literature.

The most common approach requires answers to be correct however the uncertainty represented by the database is resolved. If $\mathcal{M}$ is a set of models and Φ is a closed formula, we write $\mathcal{M} \models \Phi$ if $M \models \Phi$ for all $M \in \mathcal{M}$. Define a *certain answer* to a query $\Phi(\mathbf{x})$ on a database D to be a tuple $\mathbf{a}$ such that $Mod(D) \models \Phi(\mathbf{a})$. That is, a tuple $\mathbf{a}$ is a certain answer if $\Phi(\mathbf{a})$ holds in all models. ("Yes/No" queries Φ, in which there are no free variables, can be viewed as being included in the scope of this definition by treating the empty tuple as a certain answer when $Mod(D) \models \Phi$.)

Alternately, one could consider the tuples that *may* hold. A *possible answer* to a query $\Phi(\mathbf{x})$ is a tuple $\mathbf{a}$ such that $M \models \Phi(\mathbf{a})$ for *some* $M \in Mod(D)$. This approach appears first in the work of Lipski [Lip79]. Since a relational database D can be considered to be an incomplete database with $Mod(D)$ equal to the singleton set $\{M_D\}$, the certain answers and possible answers coincide in this case. In the more general context of incomplete databases the two notions are no longer equivalent, but possible answers are still *dual* to certain answers in the following sense: $\mathbf{a}$ is a certain answer to $\Phi(\mathbf{x})$ iff $\mathbf{a}$ is not a possible answer to $\neg\Phi(\mathbf{x})$.

Relational databases have a property, the *definite answer property*, that is not preserved by most extensions to incomplete information. Note that if M is a structure such that $M \models \exists x \Phi(x)$ then there exists a value a such that $M \models \Phi(a)$. In other words, every certain answer to an existential query can

be justified by the provision of a witness supporting this response. To see that this does not apply to an incomplete database, consider a database D such that $Mod(D)$ is the set of all first order order structures in which the formula $P(a) \vee P(b)$ is true. Then $Mod(D) \models \exists x P(x)$, but neither $Mod(D) \models P(a)$ nor $Mod(D) \models P(b)$ holds. The set of certain answers in this case is empty.

Arguably, this answer is uninformative. However, the set of possible answers is no more informative in this case. For every constant c, the model in which just $P(a)$ and $P(c)$ hold is in $Mod(D)$. Thus, the set of possible answers to the query $\{x : P(x)\}$ is the set of all constants. This again obscures the special status of a and b for this query.

Several authors have responded to these problems by proposing to return *disjunctive answers* [Rei78a; GM86; FM91; Gal86]. A disjunctive answer to the query $\Phi(\mathbf{x})$ in a database D is a set of tuples $\{\mathbf{a}_1, \ldots, \mathbf{a}_m\}$ such that $Mod(D) \models \Phi(\mathbf{a}_1) \vee \ldots \vee \Phi(\mathbf{a}_m)$. Of course, if A is a disjunctive answer, then so is any superset of A. For this reason, it suffices to return the disjunctive answers that are *minimal* under set containment. The information being returned here is *certain*, but disjunctive answers clearly have a greater expressive power than the tuples returned as certain answers. Minimal disjunctive answers have a number of interesting properties. They reveal some of the possible answers to a query, since if a tuple $\mathbf{a}$ is contained in some minimal disjunctive answer then it must be the case that $M \models \Phi(\mathbf{a})$ for some $M \in Mod(D)$. (Some possible answers may appear in no minimal disjunctive answer, e.g. b occurs in no minimal possible answer of $P(x)$ when $Mod(D) = \{\{P(a)\}, \{P(a), P(b)\}\}$. Possible answers may also be excluded because the domain is infinite.) Moreover, the set of minimal disjunctive answers to a query contains the certain answers as a distinguishable component: these are precisely the elements of the minimal disjunctive answers that are singletons. In this respect the possible answers are somewhat weaker than (minimal) disjunctive answers, as they do not make it possible to distinguish when a tuple is in fact a certain answer.

Related to disjunctive answers are approaches that have sought to develop extensions of the relational algebra. In these approaches a database consists of a set of "generalized" relations, and an answer is also of the form of such a generalized relation. This idea has the attraction of preserving the efficiency of the algebraic approach. However, many of the proposals in this area were found to be unsatisfactory once clear semantics of the generalized relations had been developed. Indeed, there are inherent limitations that prevent the obvious extensions of the relational operators with respect to most sensible logical semantics. We take up this issue in the next section.

Part	Supplier
$P33$	BigCo
$P34$	LittleCo
$P35$	LittleCo

Supplier	City
BigCo	New York
LittleCo	@
Acme	@

Figure 10.2 Codd Tables

10.4 ALGEBRAIC MODELS OF NULLS

Many of the proposals in the literature on incomplete databases have focussed on the extension of the relational model by the introduction of null values. In this section we review some of the concerns arising in these proposals. More detailed surveys of this area can be found in [AdA93; ZP96].

The earliest extension of the relational model to incomplete information was that of Codd [Cod79] who suggested that missing values should be represented in tables by placing a special *null value* symbol '@' at any table location for which the value is unknown. Figure 10.2 shows an example of a database using this convention. Codd proposed an extension to the relational algebra for tables containing such nulls, based on three valued logic and a "null substitution principle." The proposal was subsequently criticised by several authors [Gra77; Lip79] as being semantically incoherent. As Codd himself viewed his proposal as preliminary we will not reiterate the criticisms here, but will highlight some of the problems that arise in formulating a sensible extension of the relational algebra for Codd tables. (We remark, however, that the way nulls are treated in the SQL standard closely follows Codd's proposal; see Section 10.11.)

In terms of our general semantic scheme, the intended semantics of a database D consisting of Codd tables can be described by defining $Mod(D)$ to be the set of structures $M_{D'}$, where D' ranges over the relational databases obtained by replacing each occurrence of '@' in the database D by some domain value. Different values may be substituted for different occurrences. (This is not the only reasonable semantics for Codd tables: we discuss some alternatives in Section 10.7.) For example, Figure 10.1 depicts a model of the database of Figure 10.2. We also write $Mod(R)$ for the set of relations obtained by substituting values for the occurrences of '@' in the Codd table R.

To generalize the relational algebra to a class of tables $\mathcal{T}$, we need to describe how each of the operators, when applied to tables in $\mathcal{T}$, returns a result in $\mathcal{T}$. (In the special case that the tables are relations, the result should be a relation.)

A plausible constraint on the meaning of a relational operator on tables in $\mathcal{T}$ is that the result should be a table that represents the set of relations obtained by pointwise application of the operator on the models of these tables. For example, if R and S are tables in $\mathcal{T}$ then the result of the join $R \bowtie S$ should be equal to a table T in $\mathcal{T}$ such that

$$Mod(T) = \{r \bowtie s \mid r \in Mod(R),\ s \in Mod(S)\} \tag{10.1}$$

In case the definitions of the operators satisfy this constraint (with respect to the definition of the semantics *Mod* on $\mathcal{T}$), we say, following Imielinski and Lipski [IL84], that the generalized algebra is a *strong representation system*.

Let us consider what equation (10.1) requires if we take R and S to be the Codd tables in Figure 10.2. First of all, note that in each model, if we take the value of the null in the tuple (LittleCo,@) to be v, then the join will contain two tuples (P34,LittleCo,v) and (P35,LittleCo,v), both of which include the value v. If T is to be a Codd table, it will need to contain tuples (P34,LittleCo,Y) and (P35,LittleCo,X) to generate each of these tuples, where X and Y are either constants or '@'. We now face a problem. First, X cannot be a constant c, for whatever the choice of c we can find an instance $r \in Mod(R)$ and $s \in Mod(S)$ for which the tuple (P34,LittleCo,c) does not occur in $r \bowtie s$. For similar reasons Y cannot be a constant. But neither can X and Y be '@'. If they were, X and Y would have their values in models of T assigned independently. Thus, there would be instances in $Mod(T)$ containing the tuples (P34,LittleCo,v_1) and (P35,LittleCo,v_2) with $v_1 \neq v_2$, whereas in all models in the right hand side of Equation (10.1), the only tuples of this form have v_1 and v_2 equal.

What this argument shows is that Codd tables are too weak to represent the result of the join when interpreted using the semantics of Equation (10.1) and a particular function *Mod*. Maier [Mai83] has shown that the problem is more general than this: he shows that there exists no "adequate" extension of the relational algebra with respect to several other natural semantic functions *Mod*.

These considerations lead to the consideration of more expressive frameworks. The example suggests that it is necessary to relax the requirement that each null value in a table is assigned a value independently. If we allow *marked nulls*, such that repeated occurrences are intended to refer to the *same* value in each instance, it becomes possible to represent the result of the join by means

of the following table.

Part	**Supplier**	**City**
$P33$	BigCo	New York
$P34$	LittleCo	$@_1$
$P35$	LittleCo	$@_1$

Here the repetition of $@_1$ indicates that the *same* value is to be substituted for each occurrence of the null in constructing a model of the table.

Unfortunately, this extension does not suffice to satisfy the constraint (10.1). Consider the following example

Part	**Supplier**
$P32$	$@_1$

Supplier	**City**
BigCo	New York

In the model of these tables in which $@_1 = BigCo$, the join contains the tuple (P32,BigCo,New York). But if $@_1 \neq BigCo$, then the join contains no such tuple. It is not difficult to see that no table with marked nulls can represent this situation. Intuitively, what is required is an entry expressing that

$$\text{If } @_1 = \text{BigCo then (P32,BigCo,New York)} \in R \bowtie S$$

Imielinski and Lipksi [IL84] proposed to handle this difficulty by adding a column *Cond* to each table to represent a *condition* under which the tuple is true. The condition may be any boolean combination of atoms of the form *true*, *false*, $@_i = c$ or $@_i = @_j$, where c is a constant. Using this idea, the join may be represented by the table

Part	**Supplier**	**City**	**Cond**
$P32$	BigCo	New York	$@_1 = \text{BigCo}$

These considerations lead to the definition of three types of increasingly expressive tables containing null values:

1. a *Codd table* contains nulls but does not allow repetition
2. a *v-table* allows repeated nulls, but no conditional tuples
3. a *c-table* allows both repeated nulls and conditional tuples.

Imielinski and Lipski [IL84] studied the fragments of the relational algebra for which these types of tables admit a strong representation system. They show

Table Type	Strong Rep.	Weak Rep.
Codd Tables	PR	PSR
v-Tables	PUR	PS^+UJR
Horn-Tables	PS^+UJR	PS^+UJR
c-tables	PSUJRD	PSUJRD

Table 10.1 Maximal Extensions of Relational Algebra

that neither Codd tables nor v-tables admit a strong representation system for SPJUDR-queries, but c-tables do.

The conditions generated in the c-tables obtained as the result of a query may be complex. Some of these conditions may nevertheless be tautological, so that the associated tuple is a certain answer. Determining this, however, is a problem as difficult as determining validity of propositional logic formulae, i.e., coNP-complete. For this reason, c-tables may be too expressive a formalism for practical purposes. This concern motivated Imielinski and Lipski to study Codd and v-tables with respect to a weaker correctness condition.

Inasmuch as the extended types of tables express indefinite information, when we return such a table as the answer to a query, we are dealing with something akin to a disjunctive answer. The weaker condition introduced in [IL84] (and clarified in [Lip84; IL89]), relaxes this, requiring that the answer be correct only insofar as it expresses information about the certain answers to a relational query. Given a relational expression F, the certain answers are the tuples in $F(M)$ for all $M \in Mod(D)$. Thus, the result of applying the expression F to an indefinite database D should therefore satisfy

$$\bigcap Mod(F(D)) = \bigcap \{F(M) \mid M \in Mod(D)\} \tag{10.2}$$

An algebra on a class of tables $\mathcal{T}$ satisfying this condition is called a *weak representation system.* (Note that the condition is imposed for all relational expressions, not just for single applications of a relational operator.)

The maximal subsets of the relational operators for which weak and strong representation systems can be defined on the three types of tables can be constructed is described in Table 10.1. This table includes results of Grahne [Gra91], who extended the definition of c-tables to include a global condition restricting the substitutions applied to a c-table in constructing its models, and also defined *Horn tables* to be the class of such extended tables in which the global condition and the conditional tuples have the form of a Horn logic program.

Besides the models already discussed, a number of other table types have been studied. These include

1. OR-object tables, which are like v-tables except that each null has an associated finite domain [IMV95; INV91a; INV91b].
2. *Views* representing a set of worlds generated by substitutions from some table followed by the application of a query to the result [AKG91].
3. Tables with *maybe tuples*, which may or may not appear in models of the database [Bis81; Bis83; Bis84].

In some cases relational operators have been proposed for these types of tables, but not all proposals form either a weak or strong representation system. Tables containing nulls expressing that an attribute is inapplicable, or possibly inapplicable, have also been studied. The main issue being addressed by these is a mismatch of the schema to world, rather than incompleteness of information *per se*, but we discuss this area briefly in Section 10.10.1.

10.5 LOGICAL DATABASES

We introduced the model theoretic view of databases in Section 10.3. An alternative view considers a database as describing a set of facts, which may be represented as a logical theory. This approach is known as the *proof theoretic* view of databases. An important early precedent for this perspective was the work of Green [Gre69], who first considered the issue of answer extraction in theorem proving. The proof theoretic view began to draw more attention in the late 1970's [Cha76; Kow78; GM78; NG78; Rei78b], given impetus by the then emerging area of *logic programming* [Kow74], which was beginning to establish that logical deduction could be done efficiently in certain special cases.

The connection between relational databases and logical theories was made explicit in the work of Reiter [Rei78b; Rei84], who pointed out that it is possible to construe relational query processing as being equivalent to theorem proving in a precise sense. Starting with a relational database D, Reiter constructs a first order theory $T(D)$ as follows. First, we introduce a constant symbol a for each value in the database. We will ambiguously use a to denote both a value and the corresponding constant symbol. The theory $T(D)$ then contains the following formulae:

1. **Domain Closure:** The formula

$$\forall x(x = a_1 \vee \ldots \vee x = a_N)$$

where the domain of D is the set $\{a_1, \ldots, a_N\}$

2. **Unique Names:** The formula $a_i \neq a_j$ for each pair of distinct constants a_i, a_j.

3. **Equality:** The usual axioms for equality, stating that equality is an equivalence relation that satisfies the principle of substitution [Men64].

4. **Data:** For each relation R, and each tuple $\mathbf{a} \in R$, the atomic formula $R(\mathbf{a})$.

5. **Completion Axioms:** For each relation $R = \{\mathbf{a}_1, \ldots, \mathbf{a}_m\}$, the formula

$$\forall \mathbf{x}[R(\mathbf{x}) \Rightarrow \mathbf{x} = \mathbf{a}_1 \vee \ldots \vee \mathbf{x} = \mathbf{a}_m]$$

(If the relation is empty, its completion axiom is $\forall \mathbf{x} \neg R(\mathbf{x})$.)

It may then be shown that $T(D)$ is *canonical*, in the sense that its only model (up to isomorphism) is M_D. It follows that if $Q = \{\mathbf{x} : \Phi(\mathbf{x})\}$ is a relational calculus query, we have $Q(D) = \{\mathbf{a} \mid Mod(T(D)) \models \Phi(\mathbf{a})\}$. By the completeness theorem for first order logic [Men64], we may reformulate this as $Q(D) = \{\mathbf{a} \mid T(D) \vdash \Phi(\mathbf{a})\}$. That is, the problem of determining whether $\mathbf{a}$ is an answer of Q on D reduces to the problem of trying to *prove* $\Phi(\mathbf{a})$ from the theory $T(D)$.

Viewed narrowly, Reiter's logical reconstruction may seem of limited value, since it replaces the succinct tabular representation of data with a complex logical theory. Moreover, its substitute for the efficient query processing algorithms embodied in the relational algebra is the process of deduction in a first order theory. In general, not even termination guarantees can be provided for the latter. There are two benefits of the reconstruction, however. The first of these is that it makes explicit certain assumptions embodied in the relational model. One of the important observations of Reiter was that relational database theory makes a *default* assumption: the data facts are not just taken to be true, but to give a *full* description of the facts holding in the world. This assumption is made explicit by the Completion Axioms. The effect of the assumption is to make query answering *non-monotonic*: *adding* facts to a database may *decrease* the set of answers to a given query. Non-monotonicity in databases is an issue that subsequently received considerable attention. Section 10.7 discusses this area at greater length.

The second benefit of the reconstruction is that it illuminates a general framework within which generalizations of relational databases to incorporate incomplete information can be cast. Reiter proposed from this basis a specific approach to the issue of null values in relational databases, modelling these using *extended relational theories* [Rei86]. These consist of Domain Closure, Equality, Data and Completion axioms as before, but weaken the Unique Names

axiom to be a *subset* of the full set of inequalities, reflecting that the identity of some of the constants is not known. The semantics $Mod(T)$ of an extended relational theory T is simply the set of models of T. Extended Relational Theories are intermediate in expressive power between v-tables and v-tables with a global condition.

More generally, the model of incomplete databases suggested by the proof theoretic view is that a database consists of a logical theory T of some restricted form. The semantics is given by taking $Mod_{OWA}(T)$ to be all the structures satisfying T. The OWA here refers to the *Open World Assumption,* which assumes that negative information is explicitly represented in the theory T. (There are alternate approaches which build the default behaviour expressed by such axioms as the Completion Axioms directly into the semantics — we discuss these in Section 10.7.) The semantics of queries can be taken to be one of the notions of query answering discussed in Section 10.3.3. We call such an instantiation of the general scheme for indefinite databases a *logical database.*

In case the theory T is first order, there exist complete proof theories, i.e. procedures for generating the set of formulae Φ such that $Mod(T) \models \Phi$. In this case, query answering can be implemented as logical deduction using this proof theory. A disadvantage of this approach is that one cannot in general expect guarantees of termination. We now describe some of the types of logical databases that have been proposed, for which decidability of query processing can be guaranteed.

One case of note is theories containing a Domain Closure formula — in this case the remainder of the theory could be any set of sentences, and the set of queries can similarly be any set of formulae, since query processing can always be done by considering all models with domain of size at most the number of constants. In general, this is an inefficient procedure. Unfortunately, as we will see in Section 10.6, there are some inherent limitations to efficient query processing for many of the types of incomplete information that have been studied.

If the theory does not contain a Domain Closure formula, it becomes necessary to restrict the set of allowable queries. To see this, note that if we take T to be the empty theory, determining if a first order sentence Φ satisfies $Mod_{OWA}(T) \models \Phi$ amounts to determining if Φ is *valid,* an undecidable problem. A restriction that helps to achieve decidability in many cases is to take Φ to be a positive existential formula.

Many of the table types discussed in the previous section have equivalent logical databases. A careful treatment of domain closure and completion is required to construct an equivalent theory, but the positive information encoded in a table can be straightforwardly described. Codd Tables could be characterized as expressing *existential* information: a tuple such as *Part(P32,@)* can be

viewed as expressing the formula

$$\exists x \mathit{Part}(\mathit{P32}, x)$$

To represent v-tables, it is necessary to consider data axioms which are existential formulae of the form

$$\exists x_1 \ldots x_n [A_1 \wedge \ldots \wedge A_n]$$

where the A_i are atoms. A further generalization is required for c-tables, which require taking the A_i to be formulae of the form $C \Rightarrow A$, where C is a condition and A an atomic formula. For databases with OR-objects, we need to take some of the A_i to be disjunctions of the form $x = a_1 \vee \ldots \vee x = a_n$. The technique of *Skolemization* [Gal86] can be used to eliminate the existential quantifiers in these formulae. For the particular formulae under consideration, this amounts to substituting a unique constant for each occurrence of an existentially quantified variable.

All of the above, after Skolemization, are instances of *Disjunctive Logic Programs* [LMR92] which consist of rules of the form

$$\forall \mathbf{x} [(A_1 \wedge \ldots \wedge A_n) \Rightarrow (B_1 \vee \ldots \vee B_m)]$$

where the A_i and B_i are atoms. As generally understood, these may contain function symbols — the function free case, called *Disjunctive Datalog* has been considered by Eiter et al. [EGM97]. (The reason for the terminology is that logic programs consisting of function-free rules as above with $m = 1$ are called *Datalog* Programs [Ull88].)

The class of Disjunctive Datalog Programs is not the largest class of theories for which positive existential queries are known to be decidable. Imielinski [Imi91b] considers databases consisting of an *extensional* part, built from conjunction, disjunction and existential quantification, and an *intensional* part, consisting of *Skolem Rules* of the form

$$\forall \mathbf{x} \exists \mathbf{y} [(A_1 \wedge \ldots \wedge A_n) \Rightarrow (B_1 \vee \ldots \vee B_m)]$$

where the A_i and B_i are atoms. In general positive existential queries in such databases are undecidable, but a number of decidable cases are identifiable. There is a close connection between such rules and the theory of tuple generating dependencies [Kan90].

While most types of logical databases studied are first order theories, this is not essential: there are also natural classes of second order theories with decidable queries. Datalog Programs may define second order properties, such as *transitive closure*, of an underlying set of relations. The *recursively indefinite*

databases of van der Meyden [Mey93] consist of a set of definitions in Datalog and both basic and defined facts. Data of this form could arise from updates to views defined by Datalog Programs. Positive existential queries containing only extensional predicates are decidable in this context.

In addition to classes of databases falling within the scope of these cases, there has been interest in indefinite databases making use of *constraints*, such as the order constraints '$\leq$' on linearly ordered domains. We treat these separately in Section 10.10.2.

10.6 COMPLEXITY OF QUERIES

We have drawn the line between an indefinite database and a logical theory in terms of the decidability of query processing. For practical purposes, this is not necessarily the most useful demarcation: one wants query processing to be *efficient.* There is a variety of ways in which the complexity of query processing can be measured. In this section we introduce some of the measures of complexity used in the literature. We assume that the reader is familiar with the basics of computational complexity theory [Pap94].

In dealing with query complexity, it is common to restrict attention to the problem of computing answers to "Yes-No" queries, i.e., queries which are *closed* formulae Φ, rather than formulae $\Phi(\mathbf{x})$ with free variables. Thus, we are interested in being able to determine if $Mod(D) \models \Phi$ for a database D and query Φ, rather than in computing a set of answers $\mathbf{x}$ for which $\Phi(\mathbf{x})$ follows from the data. The justification for this restriction is that there exists a PTIME reduction of the problem of computing the set of certain answers $\{\mathbf{a} \mid Mod(D) \models \Phi(\mathbf{a})\}$ for the query $\Phi(\mathbf{x})$ to the "Yes-No" query answering problem. This reduction simply enumerates all the tuples $\mathbf{a}$ (of which there are d^k, where d is the number of constants in the database and k is the arity of the answer tuples), and tests $Mod(D) \models \Phi(\mathbf{a})$ for each. For disjunctive answers there is a similar reduction, but one has to face the problem that the set of answers may be exponentially large in the size of the database.

The complexity of Yes-No queries in incomplete databases can be described using a number of measures introduced for the complexity of relational database queries by Vardi [Var82]. The approach is to measure query complexity by proving both upper and lower bounds. A query problem is described as a set S, and we look for a complexity class C such that $S \in C$. This provides an upper bound for the complexity of the query problem. A lower bound can be established by proving completeness of S for the class C, i.e., showing that S is in some sense as hard as all the problems in C.

The most obvious way to represent a query problem as a set is as follows: given a class of databases $\mathcal{D}$ and a class of queries $\mathcal{Q}$, we define the answer set

$$AS_{\mathcal{D},\mathcal{Q}} = \{\langle D, \Phi\rangle \mid D \in \mathcal{D}, \Phi \in \mathcal{Q}, Mod(D) \models \Phi\}$$

In determining the complexity of this set, we ask "given as input a database D and a query Φ in the appropriate classes, how complex is it to determine if the query is entailed by the data?" Vardi [Var82] calls this measure of complexity the *combined complexity* of the query problem.

While combined complexity provides an overall view of the complexity of query processing in a class of databases, it is not always the most realistic measure. It turns out that even conjunctive queries in relational databases have NP-hard combined complexity, and hence are unlikely to be practical according to this measure! This is contrary to the fact that such databases and queries are in broad use. The explanation for this is that combined complexity uses a very poor model of the query problems that are encountered in practice.

In database applications, the size of the database can grow to be very large, as data accumulates over time. Queries, on the other hand, are composed on-line, and are therefore likely to be small compared with the size of the database. Some of the hardest instances of the query problem involve queries roughly equal in size to the data, and these instances contribute to combined complexity. Thus, combined complexity may be high only because of problem instances that are extremely unlikely to occur in practice.

A way to take the imbalance in query size and data size into account is *data complexity*, which measures the complexity of answering a fixed query as the data varies. Formally, the complexity of a particular query Φ with respect to a class of databases $\mathcal{D}$ is the complexity of the answer set $AS_{\mathcal{D}}(\Phi) = \{D \mid D \in \mathcal{D}, Mod(D) \models \Phi\}$. We may also speak of the data complexity of a class of queries $\mathcal{Q}$ with respect to a class of databases $\mathcal{D}$. This is the maximal data complexity with respect to $\mathcal{D}$ of queries in $\mathcal{Q}$.

Data complexity may be understood in a number of different ways. Most obviously, data complexity is the appropriate measure of complexity in situations in which it is necessary to ask the same query repeatedly as the data varies. Here, since the query does not change, the only contribution to complexity comes from the size of the data. In case *ad hoc* queries are of interest, these can be assumed to come from the finite set of all queries up to some fixed size. The data complexity of the infinite set of all queries will provide an upper bound on the complexity of queries from this finite set.

Although data complexity is generally a more satisfactory model of query complexity than combined complexity, in the context of incomplete information it does exhibit some peculiarities. Low data complexity does not always imply that a class of queries is practical. One reason for this is that the constants

of proportionality involved in data complexity results may be extremely large. More seriously, even if a query has a low data complexity, determining that this is the case and constructing the procedure that answers the query with this complexity may be a highly complex task.

As an example of this, take the class of databases to be the singleton set $\mathcal{D} = \{D\}$, where D is the empty theory, interpreted so that $Mod(D)$ is the set of all first order structures. Let $\mathcal{Q}$ be the set of all sentences of first order logic. Since the set of databases is a singleton, the data complexity of every query is in *constant time*: for each query Φ the constant time algorithm for answering Φ as a function of the input database is either "return(yes)" or "return(no)", depending on whether $Mod(D) \models \Phi$. However, $Mod(D) \models \Phi$ if and only if Φ is a valid formula. Since validity of first order formulae is an undecidable problem, determining which of these algorithms correctly implements the query is also *undecidable*. (A more realistic example of this difficulty occurs in [Mey97].) A number of more refined models of complexity that may go some way to addressing this problem have recently been proposed. These take into account the number of variables in the query [Var95], or are based on *parameterized complexity* [PY97]. However, the usefulness of these models in the context of incomplete databases has not yet been studied.

For completeness, we also consider the contribution to complexity due to the size of the query. The answer set of a *database* D with respect to a class $\mathcal{Q}$ of queries is the set $AS_Q(D) = \{\Phi \mid \Phi \in Q, Mod(D) \models \Phi\}$ of queries satisfied by the database. The *expression complexity* of a database is the complexity of the set $AS_Q(D)$. This is a measure of the complexity of query answering as a function of the size of the query.

The complexity of querying the types of incomplete databases discussed in Section 10.4 and Section 10.5 is in most cases very similar. Table 10.2 shows results for the complexity of positive existential first order queries in a variety of types of database. Each entry provides a complexity class for which the corresponding query problem is complete with respect to log-space transformations. In the case of data complexity, this is to be interpreted as follows:

1. For every positive existential query Φ, the data complexity of Φ is in the class indicated.

2. There exists a query Φ with data complexity complete for the class indicated. (The query can be assumed to be *conjunctive* in most cases.)

That is, it is not necessarily the case that every query has data complexity complete for the class indicated: some queries may have lower complexity. A similar interpretation applies to expression complexity. The first row

	Complexity Type		
	Data	Expression	Combined
Relational Databases	LOGSPACE	NP	NP
X-Databases	co-NP	NP	Π_2^p

Table 10.2 Complexity of positive existential queries

shows results (from [CM76; Var82]) for relational databases. We have already mentioned in motivating data complexity that combined complexity in such databases is NP-hard. Notice however, that data complexity in such databases is in LOGSPACE, so (happily) according to this measure querying relational databases is tractable. This result is an indication that data complexity is indeed a more reasonable measure of query complexity than combined complexity.

Once one moves from definite databases to databases containing even the most limited forms of indefinite information, the tractability of data complexity is lost (provided P$\neq$co-NP). The second row of Table 10.2 shows the complexity of positive existential queries in many types of incomplete databases: the expression "X-Databases" here includes Codd tables in which nulls range over a bounded domain, v-tables, c-tables, OR-object databases, logical databases containing disjunctions and Disjunctive Datalog. We refer the reader to [AKG91; Imi91b; IMV95; Var86b] for precise statements of these results in the various cases. Notice that indefinite information of these forms results in a "jump" in data complexity from LOGSPACE to co-NP, as well as a jump in combined complexity from NP to Π_2^p. While these increases are not known to be strict, it is evident that querying indefinite information is in some sense more complex than querying definite information. Whereas we have efficient algorithms for querying definite information, no such algorithms are known in the case of most sorts of indefinite information, even for the restricted class of positive existential queries. Obviously, as the expressiveness of the query language increases, the complexity can only increase. Vardi [Var86b] considers the complexity of first order and second order queries in extended relational theories.

The upper bound results in this table are not hard to explain. We assume for simplicity that $Mod(D)$ contains only models over a fixed finite domain. (This assumption can often be eliminated when dealing with positive existential queries by adapting the procedure below to inspect the set of *minimal* models

of D.) The meaning of $Mod(D) \models \Phi$ can then be directly encoded by a Co-NP computation of the form

1. Guess a model M in $Mod(D)$

2. Compute $M \models \Phi$, return *true* if so, else return *false*

where the final result is the logical conjunction of the results of all branches of the computation. The first step can be done by a linear number of choices for all the types of databases considered. The second step is NP-complete by results of Chandra and Merlin [CM76], so we obtain directly that combined complexity is in Π_2^p. The other upper bounds are derived similarly.

The lower bounds require a little more effort to prove. It is, however, worth noting that the lower bounds can be attained on extremely simple queries. To illustrate this, we show how to transform graph non-3-colorability (a well known Co-NP complete problem [GJ79]) to the data complexity of the query $\Phi = \exists xyz[E(x,y) \wedge C(x,z) \wedge C(y,z)]$ on OR-object databases. To do so, given a graph $G = \langle E, V \rangle$, we introduce an OR-object c_v, with domain $\{Red, Blue, Green\}$, for each vertex $v \in V$. In the database D, the relation E is simply the edge relation of the graph, and the relation C contains the tuple (v, c_v) for each vertex $v \in V$. Intuitively, the models in $Mod(D)$ then correspond to colorings of the graph, and the query expresses "there exist vertices x and y with the same color z." This is true in all models of the data (i.e. all colorings) if and only if the graph is non-3-colorable.

The fact that even such simple queries have high data complexity does not mean that this holds for all queries. A few special cases are known where queries have PTIME data complexity in indefinite databases. One such example is *positive* first order queries in extended relational theories [Var86b]. (It is crucial to this result that the domain in interpretations may be at least as large as the set of constants in the theory.) Imielinski et al. [IMV95] are able to give a complete characterization of the complexity of conjunctive queries in OR-databases under various assumptions concerning the occurrence of OR-objects in the data, giving syntactic conditions that separate the queries with PTIME data complexity from those with Co-NP complete data complexity. In a class of logical databases containing nulls ranging over linearly ordered domains, subject to inequalities, all positive existential queries involving only monadic relations have PTIME data complexity [Mey97]. Provided the partial order derived from the inequalities has bounded width, all positive existential queries (involving also n-ary predicates) have PTIME data complexity [MvdM92]. These special cases are quite restrictive however.

The complexity of a variety of other problems relating to incomplete databases is of interest, such as

[Membership.] Given a structure M and a database D, is $M \in Mod(D)$?

[Containment.] Given two databases D_1, D_2, is $Mod(D_1) \subseteq Mod(D_2)$?

[Consistency.] Given D, is $Mod(D)$ empty?

[Uniqueness.] Given D, is $Mod(D)$ a singleton?

A careful study of the complexity of these problems for various table types can be found in [AKG91].

10.7 NEGATIVE INFORMATION

In Section 10.5, we assumed the most obvious approach to the semantics of a logical database consisting of a theory T, taking $Mod_{OWA}(T)$ to be the set of all models of T. This is known as the *open world assumption.* While the open world assumption is natural, Reiter's logical reconstruction of relational databases shows the ubiquity of default assumptions concerning the completeness of the information in a database. Because the "negative" facts are generally far more numerous, it is often convenient to represent only "positive" information, and rely on the assumption that a fact is false if not stated explicitly. The meaning of such an assumption is clear in the context of the relational model (and more generally in the context of definite information, e.g., definite logic programs), but making it precise in the context of indefinite information is more difficult. Many different approaches have been proposed, some of which we discuss in this section. There is a large overlap between this area and the treatment of negation in logic programming (although the primary motivation there, to find semantic justification for rules of "negation by failure", is somewhat different.) Relevant survey papers are [She88; Bid91; AB94; PP90].

Before we turn to discussion of this literature, it is worth noting (following, e.g., Maier [Mai83]) that several semantics, differing in their treatment of negative facts, are reasonable for the various types of tables introduced in Section 10.4. The semantics we assumed there took the set of models to be those obtained by substituting values for each of the nulls. We will denote this semantics by $Mod_{SUB}(D)$. In the context of tables, the Open World Assumption amounts to taking $Mod_{OWA}(D)$ to contain all the models M such that each tuple of D corresponds to a fact in M, but allowing M to support any set of additional facts.

One further semantics is plausible for tables. Consider a Codd table that contains two tuples of the form $R(a, @)$ and $R(@, b)$, where @ is a null. If we apply the substitution $@ = b$ to the first tuple, then we obtain the tuple $R(a, b)$, which makes the second tuple "redundant", in the sense that there exists a substitution $(@ = a)$ under which that tuple is supported by $R(a, b)$. On the

other hand, the substitutional semantics of Codd tables will construct models containing pairs of tuples $R(a, b)$ and $R(c, b)$ with $c \neq b$. To address this, it is reasonable to consider a semantics that avoids such redundancies by restricting attention to the *minimal* models in $Mod_{OWA}(D)$. We write Mod_{MIN} for this semantics. It is apparent that $Mod_{MIN}(D) \subseteq Mod_{SUB}(D) \subseteq Mod_{OWA}(D)$.

It can be shown that if Φ is a closed positive existential formula then $Mod_{OWA}(D) \models \Phi$ iff $Mod_{SUB}(D) \models \Phi$ iff $Mod_{MIN}(D) \models \Phi$. Intuitively, this states that the default assumption embodied in the semantics preserves the positive information in the database. A similar relation to the open world semantics holds for most of the semantics for negative information that have been proposed.

Turning to approaches to negative information in logical databases, there have been numerous proposals. Some of these have been syntactic in nature, working by transforming the theory to another. Reiter's original formulation of the closure condition, called the *Closed World Assumption* [Rei78b] was of this nature. Given a theory T, he proposes capturing the default assumptions by the theory

$$CWA(T) = T \cup \{\neg A \mid A \text{ is a ground atom such that } T \nvdash A\}$$

in which all facts not entailed by T have been added. This approach suffices to give a logical reconstruction of relational databases, but it fails for indefinite databases. A simple example of this is the theory $T = \{A \vee B\}$, for which $CWA(T) = \{A \vee B, \neg A, \neg B\}$. The problem is that this theory is *inconsistent.* Similar difficulties arise in the context of extended relational theories.

The Completion Axioms discussed in Section 10.5 avoid this problem in the special case of extended relational theories. These axioms are closely related to the axioms proposed by Clark [Cla78] for the semantics of negation by failure in logic programs. The approach based on completion corresponds to the substitutional semantics Mod_{SUB} on v-tables in those cases where an extended relational theory is equivalent to a v-table.

Other closure conditions in indefinite logical databases can be viewed as related to the minimal model semantics discussed above. In general, these approaches restrict attention to Herbrand models, so embody the Unique Names and Domain Closure axioms. We may extend the definition of Mod_{MIN} above to logical databases T by taking $Mod_{MIN}(T)$ to be the set of minimal Herbrand Models of T, i.e., the set of Herbrand Models $M \in Mod_{OWA}(T)$ such that if $M' \in Mod_{OWA}(T)$ and $M' \subseteq M$ then $M = M'$. An attempt to formulate this semantics syntactically in the case of disjunctive logic programs is the *generalized closed world assumption* of Minker [Min82], which replaces a disjunctive logic program P by

$$GCWA(P) = P \cup \{\neg A \mid A \text{ is a ground atom such that } Mod_{MIN}(P) \models \neg A\}$$

It turns out that $GCWA(P)$ does not express $Mod_{MIN}(P)$. Instead, to express this set of models requires the *Extended Generalized Closed World Assumption* of Yahya and Henschen [YH85], defined as

$$EGCWA(P) = P \cup \{\neg C \mid \begin{array}{l} C = (A_1 \wedge \ldots \wedge A_n) \text{ is a finite conjunction of} \\ \text{ground atoms } A_i \text{ such that } Mod_{MIN}(P) \models \neg C\}. \end{array}$$

It can then be shown that $Mod_{MIN}(P)$ is equal to the set of Herbrand models in $Mod_{OWA}(EGCWA(P))$.

There are also semantics of disjunctive logic programs that are somewhat like the substitutional semantics Mod_{SUB}, in that they define sets of models between $Mod_{MIN}(P)$ and $Mod_{OWA}(P)$. In the context of disjunctive information, the motivation for this is to interpret disjunction non-exclusively. For example, one might wish the Herbrand models of the database $\{A \vee B\}$ to be $\{A\}$, $\{B\}$ and $\{A, B\}$. Proposals to this end are the Disjunctive Database Rule of Ross and Topor [RT88] and the equivalent Weak Generalized Closed World Assumption [RLM89]. It has been argued that integrity constraints are not properly treated in this semantics, leading to (equivalent) refinements [Cha93; Sak89].

In addition to these proposals, there is a host of semantics that seek to generalize logic programming by accommodating versions of negation as failure [Llo87]. The literature in this area is too extensive to cover here in any detail. Prominent in the context of non-disjunctive logic programs are *stable model* semantics [GL88], the *well-founded model* semantics [GRS91], and a fixpoint semantics for stratified programs [ABW89]. (See Chapter 2 for a definition of these semantics.) There have been extensions of each proposed for disjunctive logic programs [Prz89; Prz91; Prz95]. There are also approaches that seek to develop frameworks combining classical negation and negation as failure [GL91; Prz91]. The relation of such semantics to the large variety of methods for *non-monotonic reasoning* developed in the Artificial Intelligence literature has also been a topic of considerable interest [Prz93].

The semantics discussed above share the property that they seek to formalize the default that *all* facts are false unless stated otherwise. A number of authors have proposed mechanisms whereby the application of this default may be limited to selected facts or relations [MP85; MP84; GZ88]. Motro [Mot89] argues that in the context of partially closed databases, an answer should be accompanied by a description of its integrity, indicating whether the answer is known to be complete, and suggests an approach to determining completeness based on query rewriting. An alternate approach to determining answer completeness is presented in [Lev96].

Complexity results for query processing and consistency checking of *propositional* disjunctive logic programs under the semantics discussed above, and

others, are surveyed in [EG95]. In general, determining whether a ground atom A holds in the set of all models of a program defined by these semantics is Π_2^p-complete. An exception of note is that this problem is in PTIME for the Disjunctive Database Rule and its extensions [Cha93]. (However, in general, positive existential queries still have CoNP-hard data complexity in this context.) Complexity and expressiveness of disjunctive Datalog programs under minimal, perfect and stable model semantics is treated in [EGM97].

10.8 INTEGRITY CONSTRAINTS

An important aspect of the relational model is its treatment of *integrity constraints*. For a given application, not all relations for a schema are sensible. For example, each social security number should identify a unique person, so a relation R with attributes *Person* and *SSN* containing tuples (Jones,111-222-333) and (Smith,111-222-333) is unacceptable. This constraint is an example of a *functional dependency*, the most common type of constraint. The study of integrity constraints in the relational model has led to the development of a rich body of research dealing with a variety of constraints and their inference problem: see [Kan90; GGGM97] for recent surveys.

Integrity constraints can be presented as formulae of first order logic. (A richer framework for the statement of integrity constraints, making use of modal logic, has been proposed by Reiter — we discuss this in Section 10.10.6.) For example, the functional dependency above corresponds to the formula

$$\forall y_1 y_2 x[R(y_1, x) \wedge R(y_2, x) \Rightarrow y_1 = y_2]$$

Many of the types of dependencies that have been studied have a similar syntactic structure: being rules of the form

$$\forall \mathbf{x} \exists \mathbf{y}[B \Rightarrow H]$$

where B and H are conjunctions of atoms. A relational database D is said to *satisfy* a dependency Ψ expressed in first order logic if its model M_D satisfies Ψ. Satisfaction of the dependencies provides a way of checking that the data are not incoherent.

A number of ways have been proposed to generalize this definition to incomplete databases. In the *consistency* approach [Kow78] an incomplete database D satisfies an integrity constraint Ψ if there exists a model $M \in Mod(D)$ such that $M \models \Psi$. The *entailment* approach to dependency satisfaction [Rei84] requires that $M \models \Psi$ for *all* $M \in Mod(D)$. Note that the consistency definition of satisfaction is not *additive*: a database can be consistent with constraints Ψ_1 and Ψ_2 without being consistent with the conjunction $\Psi_1 \wedge \Psi_2$. The appropriate way to apply this definition is therefore to the complete *set* of integrity constraints.

The distinction between the two definitions blurs somewhat once one considers query processing in the presence of integrity constraints. In relational databases, integrity constraints satisfied by a database may be ignored during query processing (although they could be used to *optimize* queries). However, ignoring integrity constraints that are satisfied by an indefinite database under the consistency definition may lead to undesirable results. Consider the v-table

Part	**Supplier**	**City**
$P33$	Acme	New York
$P34$	Acme	$@_1$

with the functional dependency *Supplier* $\rightarrow$ *City*, interpreted under the substitutional semantics Mod_{SUB}. Let Q be the query "Which parts are supplied from New York?" Note that the integrity constraint is satisfied in exactly one model of the database, that in which $@_1$ ="New York". All other models fail to satisfy the constraint. Thus, the most reasonable certain answer to the query is {P33,P34}, not {P33} as we would obtain as certain answer on the basis of $Mod_{SUB}(D)$.

It appears from this that the appropriate approach to answering queries in an incomplete database D in the context of integrity constraints Ψ is to work with respect to the set of models $\{M \in Mod(D) \mid M \models \Psi\}$. This is tantamount to viewing the dependency as part of the data itself. Some types of incomplete database can accommodate many types of integrity constraint directly. For example, in (disjunctive) logic programs it is common to allow the database to contain constraints of the form

$$\forall \mathbf{x}[B(\mathbf{x}) \Rightarrow \mathbf{false}]$$

where B is a conjunction of atoms [LT85; LMR92]. In general, however, a problem with the incorporation of integrity constraints in the database is that they prevent the use of special query processing procedures adapted to the restricted syntax of the database itself, because they are formulated using different syntax.

This leads to the question of whether the database can be transformed into a state that enables the integrity constraints to be ignored during query processing. A number of authors have studied techniques whereby this may be done for databases with null values [Gra91; Imi91a; Ler86; Vas80]. A problem arises that is similar to that discussed above in the context of generalizing the relational algebra to tables: in general, the set of models $\{M \in Mod(D) \mid M \models \Psi\}$ cannot always be expressed by a table. If it can, we say that the class of tables forms a *strong dependency system* for the class of integrity constraints under consideration.

A weaker notion can be considered [IL83] that is more often attainable. Say that a set of databases $\mathcal{D}$ forms a *weak dependency system* with respect to a class of dependencies Γ and queries $\mathcal{Q}$ if for all $D \in \mathcal{D}$ and $\Psi \in \Gamma$ there exists $D' \in \mathcal{D}$ such that for all $Q \in \mathcal{Q}$ we have

$$\bigcap\{Q(M) \mid M \in Mod(D) \text{ and } M \models \Psi\} = \bigcap\{Q(M) \mid M \in Mod(D')\}$$

That is, the certain answers to queries in $\mathcal{Q}$ cannot distinguish between the the models of D satisfying the integrity constraints and the set of all models of D'. See [Gra91; Imi91a] for results concerning weak and strong dependency systems.

10.9 UPDATES OF INCOMPLETE DATABASES

There are a number of different perspectives from which the issue of updates in incomplete databases can be addressed, and there are close connections to problems arising already for updates of relational databases.

In the model theoretic view of relational databases, performing an update operation can be considered as the application of a function transforming models of the schema. That is, writing $Mod(\mathcal{S})$ for the class of all (finite) models conforming to a schema $\mathcal{S}$, an update operation semantically corresponds to a function $U : Mod(\mathcal{S}) \to Mod(\mathcal{S})$. For example, the meaning of an update expression such as "insert($R(t)$)" is the function that adds the tuple t to the relation R, keeping the remaining relations constant. More general types of update function could be considered on this view, such as "raise the salary of all managers by 10%", though it is common to view these as defined over a basis of the primitive operations such as *insert* and *delete*. The effect of an update operation U on a database D can then be taken to be the substitution of the model M_D by the model $U(M_D)$.

It is proposed in [AG85] to apply this perspective on updates to the more general context of incomplete databases by applying the update operation pointwise to the models of the database. That is, the semantics of an update operation is still a function U as above, but it now transforms the *set* of models $Mod(D)$ to the set $U(Mod(D)) = \{U(M) \mid M \in Mod(D)\}$. We see that the question of expressiveness arises again: does there exist a database D' such that $Mod(D') = U(Mod(D))$? In general, there does not, and it is necessary to consider a weaker notion similar to that discussed above for algebraic operations and integrity constraints. Results on this issue are treated in [Gra91].

An alternate, more declarative, understanding of update in relational databases is to take the meaning of an operation such as "insert($P(t)$)" to be "make the minimal change to the database required to ensure that $P(t)$ is true". This perspective leads to a somewhat different approach to update

databases. It suggests generalizing the primitive update operations to take as argument not just a tuple, but any *formula*, so that we may state update operations such as, e.g., "insert($\forall x \exists y[SSN(x, y)]$)" to enforce the constraint that all persons should have a social security number.

How to make sense of this idea is a problem that arises also in the context of the *view update problem* in relational databases. A *view* of a relational database D is defined by a query Q (or more generally, a set of queries). Instead of interacting with the database D directly, the user interacts with the answer $Q(D)$. The reasons for restricting the user in this way are varied: the user may have interests limited to the data present in the view, may be prohibited from accessing other parts of the database, or the view may represent the part of a distributed database that is stored locally on the user's machine.

A general principle that may be applied to a view is that the user should be able to operate on it as if it were the database itself. This leads to the desideratum that a user should be able to update a view just as she would a database. How to implement this principle is problematic, however. Since in each state of the database the view is supposed to correspond to the result of applying the query Q, to effect an update u on a view $Q(D)$ we need to find a database instance D' such that $u(Q(D)) = Q(D')$. In general, how to do so is unclear, since there may be *many* D' satisfying this equation. One would prefer to make some *minimal* change to D in producing D', but even this restriction is not sufficiently clear to determine a unique candidate, and many approaches have been proposed. We refer to [FC85] for a survey of the literature on view updates.

One possible response to the existence of many candidates for the result of updating a relational database is to represent the result of a view update as an *incomplete database*, an approach that is most clearly sensible in the case of insertions into views defined as projections (cf. the discussion in Section 10.2). In general, however, this approach raises as many questions as it answers, since a sequence of two updates applied to a database now requires that for the second we describe how an update applies to an *incomplete* database. Should updates in logical databases operate syntactically, being transformations of the theory, or should they be defined semantically, as transformations of sets of models? What is the meaning of a "minimal change" to a set of models of a theory, or of a theory itself? If we define the semantics of an update to be a transformation on sets of models, should such a transformation operate pointwise or on the input set as a whole?

Many different, but apparently reasonable, answers have been given to these questions [AT89; DvdR92; Esc90; FKUV86; FUV83; Heg87; KW84; KW85; Osb81; Win90]. Given the difficulty of justifying the use of one approach over another in this area, it has become common to advert to general principles that

any theory of update should satisfy, and to verify that the particular model proposed in fact satisfies those principles. Influential in this area have been the "AGM" principles proposed by the philosophers Alchouron, Gardenfors and Makinson in the context of epistemology [AGM85], which have also been the subject of considerable attention in Artificial Intelligence. The resulting literature is too extensive to treat in detail: we refer the reader to the recent survey [GR95]. We note just a few works of specific relevance in the context of incomplete databases, arising in the work of researchers motivated primarily by database applications.

One point noted by database researchers is that the AGM principles do not adequately capture updates resulting from a change in the state of the world, and that an alternate set of principles should be applied in this case [KM92]. Revesz [Rev93] argues that another set of principles should be applied in contexts where it is necessary to arbitrate between different sources of information.

There is a close connection (noted, e.g., in [GM95]) between updates and *hypothetical* queries, which are queries such as "if Charles were to become king, would the monarchy be abolished?" A hypothetical query of the form "if φ were to hold, would ψ be the case?" may be implemented by what is known in the literature as the *Ramsey test*: update the database by φ, and then pose the query ψ. Generalizing earlier work on hypothetical queries in definite databases, a framework in which complex updates of incomplete databases may be defined over a basis of more primitive update operations has been proposed in [BK94; BK97].

The literature has also been characterized by a focus on the complexity of approaches to updates of incomplete databases. The complexity of hypothetical queries is studied in [EG92; EG96]. In general, approaches based on minimal change are intractable, irrespective of whether the semantics is based on theory or model based operations. The one exception to this rule is revisions of Horn theories, but this restriction excludes indefinite databases. The positive side of these negative results, however, is that the high complexity makes update a highly expressive framework for the formulation of queries [GMR97].

The complexity of performing the update itself is subject to an interesting tradeoff. One could make the cost of an update of a database D by a formula φ be proportional to the size of φ simply by representing the revised database as "D updated by φ". This has the disadvantage of raising the cost of subsequent queries. On the other hand, to transform the result of the update to a database in a more standard form (a table, say), may be a complex operation, and much of the work may be wasted if no subsequent query relevant to φ is posed. A detailed study of these tradeoffs for a particular semantics of update has been carried out by Winslett [Win90].

10.10 OTHER ISSUES

10.10.1 Inapplicable Attributes

In some cases in which the user is unable to supply a value for the attribute of some tuple in a relational database, the reason this is so is that the attribute is *inapplicable* for the object in question. For example, while a relation may have an attribute for the middle name of a person, not everyone has a middle name. To place a null in the middle name attribute, indicating that the name is unknown, would be misleading. Rather, the problem is a mismatch between the schema and the world being modelled.

It may nevertheless be appropriate to view the situation as so exceptional that it does not warrant complicating the schema, and place a special marker in the inapplicable attribute. Such a marker may be called a *nonexistent* null. Nulls of this form are studied in [Vas79; Vas80; LL86]. The *no information null* [AM84; Zan84; Kel86] is a hybrid expressing that either the attribute is inapplicable or its value is unknown. Such nulls are crucial to a satisfactory characterization of Weak Instances, a construct that enables normalized databases to be queried through the schema used in design. We refer to [AdA93] for a discussion of this topic.

The complexity of integrity constraint satisfiability for a model of incomplete databases related to the Universal Relation Assumption has been studied by Vardi [Var86a].

10.10.2 Constraints

Certain relations, such as temporal precedence or linear order on the integers, are sufficiently common in applications that it is desirable that data models provide built-in support for them. This is particularly the case in applications such as scheduling and design. In the context of definite data, this has motivated areas of research such as *temporal data models* [TCG$^+$93], *constraint logic programming* [JL87] and *constraint databases* [KKR95]. There have been a number of works dealing with constraints in the context of indefinite databases.

Temporal database models capable of representing indefinite temporal data have been proposed in [DS93; GNP92]. The complexity of query processing in (open world) databases consisting of a v-table together with a global constraint comprised of a conjunction of statements of the form $x < y$ or $x \leq y$ is studied in [Mey97].

Constraint databases [KKR95] represent a type of definite data such as that expressed by the formula

$$\forall x[x > 3 \Rightarrow P(a, x)]$$

which implies that an infinite number of facts hold in the relation P. The combination of indefinite databases and such infinite information has been investigated in [SRR94; Kou94a; Kou97]. Algebraic query languages for such databases are investigated in [Kou94b].

10.10.3 Object Oriented Databases

A database is *non-first normal form* if the values of some attributes of a tuple may themselves be relations. Null values and extensions of the relational algebra in the context of such databases are discussed in [RKS89; RKS91; LL91; Lev92]. An interesting issue arises in the interpretation of embedded empty relations in this context: it has been argued that this is akin to a null value. Related issues in Object-Oriented databases are discussed in [Zic90], which also identifies a new form of incompleteness that arises in this setting: incompleteness of the specification of attributes and methods applying to a class.

Buneman, Jung and Ohori [BJO91] proposed to study incomplete database objects using tools from domain theory. This has lead to the consideration of a number of powerdomain orderings and a variety of approximations appropriate to incomplete complex objects [JLP92; JP95; LL90; Lib91; Oho90b; Oho90a]. A careful comparison of the orderings considered by these authors is presented by Libkin in his thesis [Lib94], where he provides a justification of each ordering by showing it can be obtained from sequences of "information-increasing" update operations. Libkin also identifies category-theoretically "natural" programming constructs based on the corresponding approximations, but finds them too complex for practical use. Libkin and Wong [LW96] have defined a nested data model that generalizes OR-objects. The complex objects in this model can be normalized so that they contain a single level of the OR construct [Lib95]. This model has been been realized in an extension to the functional programming language SML called OR_SML [LG94].

Related to object oriented models are *Description logics* (also known as *terminological logics*), which are languages for describing classes of objects in terms of constraints on their attributes. For example, the expression

$$\mathbf{and}(\mathit{COURSE}, \mathbf{all}(\mathit{students}, \mathit{CS\text{-}major}), \mathbf{at-least}(7, \mathit{students}))$$

denotes the class of courses with at least 7 students enrolled, all of whom are are Computer Science majors. Borgida [Bor95] argues that description logics can be interpreted as representing a type of incomplete information, and that this fact underlies the usefulness, for design applications such as configuration management, of systems which compute subsumption relationships between such expressions.

10.10.4 Design of indefinite databases

Whereas schema design and normalization have been the subject of considerable attention for relational databases, very little work has been done in this area for incomplete databases. Codd [Cod79] states the restriction that nulls should not appear in the primary key of a relation. Maier [Mai80; Mai83] defines *existence constraints*, a type of integrity constraint that provides a way to express such restrictions. A number of papers [AM86; Gol81] consider inference rules for such constraints, and show that they are related to the notion of *objects* in the Universal Instance Assumption [Sci80].

Imielinski et al. [IMV95] propose an approach to schema design for OR-object databases called Complexity Tailored Design. The approach assumes that a fixed finite set Q of queries can be determined at design time, for which efficiency guarantees must be provided. Based on a comprehensive analysis of the data complexity of these queries as a function of the setting of design parameters relating to the distribution of OR-objects in the database, a "maximal" setting of these parameters can be determined that allow the queries in Q to be answered with PTIME data complexity.

10.10.5 Dealing with Query Complexity

The complexity results for query processing discussed in Section 10.6 indicate that obtaining complete sets of answers is likely to be intractable for most types of indefinite database. There have been a number of ways proposed to deal with this inherent intractability: *approximations* to the answer set, and procedures to reduce the degree of incompleteness in the database.

A procedure for approximate query processing in extended relational theories T has been proposed by Reiter [Rei86], and generalized by Vardi [Var86b]. The idea of the procedure is to first convert formula to a normal form with all negations pushed to atomic formulae, using a well-known transformation. This leaves a query containing negated atoms and negated equations. The idea of the procedure is to replace these by an approximation: for example, a unary atom $\neg P(x)$ is replaced by "for all constants a such that $P(a)$ is in T, the atom '$x \neq a$' is in T." In effect, this checks that the atom $\neg P(x)$ is *a logical consequence* of T. The treatment of equations and n-ary negated atoms is similar: the latter involves the identification of a computable condition on tuples $\mathbf{a}, \mathbf{b}$ that is equivalent to $\mathbf{a} \neq \mathbf{b}$ being a logical consequence of T. Reiter describes this condition by reference to an equivalence relation on the constants in the database; Vardi shows that it can be expressed as a formula. This procedure always returns a subset of the set of all certain answers. It can be shown that it returns the complete set of answers in a number of circumstances: if the database is complete, or if the query is positive.

One of the motivations for the study of *non-monotonic logics* [Gab94] in the Artificial Intelligence literature has been to reduce the complexity of dealing with incomplete information. Paradoxically, however, in most proposals the cost of eliminating incompleteness is greater than the benefit obtained in reduced query complexity. Limited use of defaults may still be efficiently implementable, however. Royer [Roy91] proposes that indefiniteness in logical databases be reduced by the application of a preference order on ground atoms. For example, if the atom A is preferred to B, a disjunction $A \vee B$ would be replaced by the A. Date [Dat86] argues that query processing in relational databases with nulls should be disallowed, but that attributes in a relational database accepting nulls should be accompanied with a default value to be substituted in case no value is supplied by the user for a particular tuple.

10.10.6 Modal and Non-standard Logics

It was noted by Lipski [Lip79] that the distinction between certain and possible answers can be characterized using modal logic. If φ is a formula, define the meaning of the formula $K\varphi$ at a model M of a database D by

$$D, M \models K\varphi \text{ if } D, M' \models \varphi \text{ for all } M' \in \mathit{Mod}(D).$$

Intuitively, the operator K expresses the modality "it is known that', or "it is necessarily the case that". For a database D, we define $D \models \varphi$ to mean that $D, M \models \varphi$ for all $M \in \mathit{Mod}(D)$. This definition is an example of a more general approach to the semantics of modal logic using what are known as *Kripke structures* [Che80]. The set of certain answers to a query $\Phi(\mathbf{x})$ can then be characterized as the set $\{\mathbf{a} \mid D \models K\Phi(\mathbf{a})\}$, and the set of possible answers can be characterized as the the set $\{\mathbf{a} \mid D \models \neg K\neg\Phi(\mathbf{a})\}$. Intuitively, $\neg K\neg\varphi$ expresses that φ is *possible* or *consistent with what is known.*

Levesque [Lev84] observes that the semantics of K just presented captures a notion of knowledge that corresponds to the contents of the database D capturing "all that is known". He shows that the operator K can be used to express queries about the degree of completeness of the database. For example, the query

$$\neg K(\exists x[\text{Teacher}(x) \wedge \neg K\text{Teacher}(x)])$$

holds in D under the semantics above if the database does not know if it has a complete information concerning the set of teachers.

Similar observations lead Reiter [Rei88] to propose that that integrity constraints in incomplete databases should be expressed in modal logic. He argues that neither the consistency nor entailment definitions satisfactorily capture the required constraints. The constraint that every employee listed in the database

should have a social security number would be expressed as

$$\forall x[K\text{Employee}(x) \Rightarrow \exists y K\text{SS\#}(x, y)]$$

on this proposal.

One can take these considerations further and include modal assertions themselves in the database. This leads to the topic of *auto-epistemic* logic [Moo85; MT91]. Databases containing assertions about their own knowledge can support patterns of reasoning such as "If I had a brother I would know about it. I don't know if I have a brother, so I must not have one." In this argument, incompleteness of knowledge is used to make knowledge more complete. The topic of auto-epistemic reasoning has been shown to be closely related to the semantics of negation in logic programs, and there have been frameworks proposed incorporating auto-epistemic operators in disjunctive logic programs [Gel94].

Other works applying modal logic to incomplete databases are [Kwa91] and [Lip81; Ost87]. The latter consider axioms and restrictions on Kripke structures derived from an "increasing information" order on indefinite databases. This order yields a class of structures that are closely related to the Kripke semantics for intuitionistic logic. Dong and Lashmanan [DL94] formulate conditional answers (related to c-tables) within a framework of logical databases of the form of logic programs with embedded implications, which have an intuitionistic interpretation.

There have also been approaches to incomplete databases based on other types of nonstandard logics, including 3 valued logic [Cod79; JFM92; YFM92; Yue91], 4 valued logic [Ges90; Ges91], and paraconsistent logic [Sub90].

10.11 INCOMPLETE INFORMATION IN CURRENT TECHNOLOGY

We conclude by briefly discussing the approach taken to indefinite information in commercial relational database systems. In general, commercial systems seek to conform to the SQL standard, which has undergone a number of revisions and continues to be further developed. Throughout the discussion, we refer to the SQL'92 standard, described in [DD93].

SQL provides for null values in a way that amounts essentially to Codd tables. In SQL, a relation scheme may be defined by means of an expression such as

```
CREATE SUPPLIER
   (SNO = CHAR(5) NOT NULL
    SNAME = CHAR(20) DEFAULT '*name unknown*')
```

The effect of a `NOT NULL` declaration on a column is that that null values are never admitted into this column. Columns in the primary key of a relation

are always interpreted as being declared `NOT NULL`. The effect of a `DEFAULT` declaration is to replace any unspecified value in a tuple added to the relation `S` with the default value specified for the column. The default value may be a specific value of the appropriate type, NULL or USER. The latter sets the value of an unspecified column to be the name of the user responsible for the update introducing the tuple.

INGRES [Sto86; Dat87] allows for a slightly different use of default values, as a *replacement* for nulls, along the lines of Date's proposal [Dat86]. An INGRES schema may contain a `NOT NULL WITH DEFAULT` declaration on a column. The effect of such a declaration is to replace any null value in a tuple added to the relation `S` with the default value appropriate for the type of the corresponding column. For numeric types this is zero; for fixed length strings it is the string of blanks of the appropriate length; for variable length strings it is the empty string.

SQL queries are evaluated using three valued logic, following Codd [Cod79]. Comparisons involving a null value always return the truth value "unknown". However, some SQL operators, including "EXISTS" return only two values. (Date [Dat89] illustrates some counter-intuitive consequences resulting from the SQL interpretation of the EXISTS operator.) The inconsistency is compounded by the treatment of relational operators such as union. For example, the union of two relations, each containing the tuple `(Acme, @)`, contains a single copy of the tuple. In effect, this amounts to the assumption that the values being represented by the two nulls in these tuples is the same. This violates the principle that a comparison of nulls should evaluate to "unknown".

SQL also provides for a quite expressive set of types of integrity constraints, but the interaction of these constraints with null values is weak, from a logical point of view. No attempt is made to integrate integrity constraints with query processing along the lines discussed in Section 10.8. The primary interaction is in the way integrity constraints limit the occurrence of null values. For example, columns, or sets of columns, of a relation may be specified to be UNIQUE, which constrains the relation to contain at most one tuple with any given value(s) those columns. Columns specified to be unique implicitly acquire the NOT NULL constraint, so may not contain null values.

The inconsistencies in the SQL standard mean that it is not possible to ascribe any intuitive logical semantics to the treatment of nulls in SQL. This is not surprising: we have already noted in Section 10.4 that it is a fundamental limitation of Codd tables that the set of relational operators for which they form a (weak or strong) representation system is severely restricted.

Unfortunately, the inherent computational complexity of query processing in semantically more satisfactory proposals has thus far prevented these from having much impact on database practice. This may change as the technology

is increasingly applied to applications, such as planning and design, which are more demanding of support for handling of incomplete information, and which may be prepared to pay the cost of high query complexity.

References

[AB94] K. R. Apt and R. N. Bol. Logic programming and negation: A survey. *Journal of Logic Programming*, 19/20:9–71, 1994.

[ABW89] K.R. Apt, H.A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufman, 1989.

[AdA93] P. Atzeni and V. de Antonellis. *Relational Database Theory*. Benjamin/Cummings, Redwood City, CA, 1993.

[AG85] S. Abiteboul and G. Grahne. Update semantics for incomplete databases. In *Proc. Int. Conf. on Very Large Data Bases*,, pages 1–12, Stockholm, Sweden, August 1985.

[AGM85] D. Alchouron, P. Gardenfors, and D. Makinson. On the logic of theory change: partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[AKG91] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78:159–187, 1991.

[AM84] P. Atzeni and N. M. Morfuni. Functional dependencies in relations with null values. *Information Processing Letters*, 18(4):233–238, 14 May 1984.

[AM86] P. Atzeni and N. M. Morfuni. Functional dependencies and constraints on null values in database relations. *Information and Control*, 70(1):1–31, July 1986.

[ANS75] ANSI/X3/SPARC. Study group on database management systems. *SIGMOD FDT Bulletin*, 7(2), 1975.

[AT89] P. Atzeni and R. Torlone. Approaches to updates over weak instances. In *Proc. First Symposium on Mathematical Fundamentals of Database Systems*, pages 12–23, Visegrad, Hungary, June 1989.

[Bid91] N. Bidoit. Negation in rule-based database languages: a survey. *Theoretical Computer Science*, 78(1):3–83, 21 January 1991.

[Bis81] J. Biskup. A formal approach to null values in database relations. In H. Gallaire, J. Minker, and J. Nicolas, editors, *Advances in Data Base Theory*, volume 1, pages 299–341. Plenum Press, New York, 1981.

[Bis83] J. Biskup. A foundation of Codd's relational maybe-operations. *ACM Transactions on Database Systems*, 8(4):608–636, December 1983.

[Bis84] J. Biskup. Extending the relational algebra for relations with maybe tuples and existential and universal null values. *Fundamenta Informaticæ*, VII(1):129–150, 1984.

[BJO91] P. Buneman, A. Jung, and A. Ohori. Using powerdomains to generalize relational databases. *Theoretical Computer Science*, 91(1):23–55, 9 December 1991.

[BK94] A. J. Bonner and M. Kifer. An overview of transaction logic. *Theoretical Computer Science*, 133(2):205–265, 1994.

[BK97] A. Bonner and M. Kifer. Logic Programming for Database Transactions. chapter 5. 1997.

[Bor95] A. Borgida. Description logics in data management. *IEEE Transactions on Knowledge and Data Management*, 7(5):671–68, 1995.

[Cha76] C. L. Chang. DEDUCE: A deductive query language for relational data bases. In C. H. Chen, editor, *Pattern Recognition and Artificial Intelligence*, pages 108–134. Academic Press, New York, 1976.

[Cha93] E. Chan. A possible world semantics for disjunctive databases. *IEEE Transactions on Data and Knowledge Engineering*, 5(2):282–292, 1993.

[Che80] B. Chellas. *Modal Logic*. Cambridge University Press, 1980.

[Cla78] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[CM76] A.K. Chandra and P.K. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 77–90. Association for Computing Machinery, 1976.

[Cod70] E. F. Codd. A relational model for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[Cod72] E. F. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Data Base Systems*, pages 33–64. Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[Cod79] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, December 1979.

[Dat86] C. J. Date. *Relational Database: Selected Writings*, chapter 15: Null Values in Database Management, pages 313–334. Addison-Wesley, Reading, MA, 1986.

[Dat87] C. J. Date. *A guide to INGRES : a user's guide to the INGRES product*. Addison-Wesley, Reading, Mass, 1987.

[Dat89] C. J. Date. *Relational Database Writings 1985 - 1989*, chapter 13: EXISTS is Not 'Exists'! (some logical flaws in SQL), pages 339–356. Addison-Wesley, Reading, MA, 1989.

[DD93] C. J. Date and H. Darwen. *A Guide to The SQL Standard, 3rd ed.* Addison-Wesley, Reading, MA, 1993.

[DL94] F. Dong and L. V. S. Lakshmanan. Intuitionistic interpretation of deductive databases with incomplete information. *Theoretical Computer Science*, 133(2):267–306, 1994.

[DS93] C. E. Dyreson and R. T. Snodgrass. Valid-time indeterminacy. In *Proceedings of the International Conference on Data Engineering*, pages 335–343, Vienna, Austria, April 1993.

[DvdR92] F. Dignum and R. P. van de Riet. Addition and removal of information for a knowledge base with incomplete information. *Data & Knowledge Engineering*, 8:293–307, 1992.

[EG92] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 57(2-3):227–270, 1992.

[EG95] T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15:289–323, 1995.

[EG96] T. Eiter and G. Gottlob. The complexity of nested counterfactuals and iterated knowledge base revisions. *Journal of Computer and System Sciences*, 53(3):497–512, 1996.

[EGM97] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.

[Esc90] C. Esculier. Non-monotonic knowledge evolution in VLKDBs. In *Proc. of the 16th Int. Conf. on Very Large Databases*, Brisbane, Australia, 1990.

[FC85] A.L. Furtado and M.A. Casanova. Updating relational views. In W. Kim, D. Reiner, and D. Batory, editors, *Query Processing in Database Systems*. Springer-Verlag, 1985.

[FKUV86] R. Fagin, G. Kuper, J. D. Ullman, and M. Y. Vardi. Updating logical databases. In *Advances in Computing Research*, volume 3, pages 1–18, 1986.

[FM91] J.A. Fernandez and J. Minker. Bottom-up evaluation of hierarchical disjunctive deductive databases. In K. Furukawa, editor, *Logic Programming: Proceedings of the Eighth International Conference*, pages 660–675. The MIT Press, 1991.

[FUV83] R. Fagin, J. D. Ullman, and M. Y. Vardi. On the semantics of updates in databases. In *Proc. ACM Symp. on Principles of Database Systems*, pages 352–365, 1983.

[Gab94] D. M. Gabbay, editor. *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume III: Nonmonotonic Reasoning and Uncertain Reasoning. Oxford University Press, Oxford, 1994.

[Gal86] J. Gallier. *Logic for Computer Science: Foundations for Automatic Theorem Proving*, chapter 9: SLD-Resolution and Logic Programming. Computer Science Series. Harper and Row, New York, 1986.

[Gel94] M. Gelfond. Logic programming and reasoning with incomplete information. *Annals of Mathematics and Artificial Intelligence*, 12:89–116, 1994.

[Ges90] G. H. Gessert. Four value logic for relational database systems. *SIGMOD Record*, 19(1):29–35, 1990.

[Ges91] G. H. Gessert. Handling missing data by using stored truth values. *SIGMOD Record*, 20(3):30–42, September 1991.

[GGGM97] P. Godfrey, J. Grant, J. Gryz, and J. Minker. Integrity Constraints: Semantics and Applications. chapter 9. 1997.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R.A. Kowalski and K.A. Bowen, editors, *Logic Programming: Proc. Fifth International Conference and Symposium*, pages 1070–1080. The MIT Press, 1988.

[GL91] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[GM78] H. Gallaire and J. Minker. *Logic and Data Bases.* Plenum Press, New York, 1978.

[GM86] J. Grant and J. Minker. Answering queries in indefinite databases and the null value problem. In P. Kanellakis, editor, *Advances in Computing Research*, volume 3, pages 247–267. JAI Press, London, 1986.

[GM95] G. Grahne and A. O. Mendelzon. Updates and subjunctive queries. *Information and Computation*, 116:241–252, 1995.

[GMR97] G. Grahne, A. O. Mendelzon, and P. Z. Revesz. Knowledgebase transformations. *Journal of Computer and System Sciences*, 54:98–112, 1997.

[GNP92] S. K. Gadia, S. S. Nair, and Y.-C. Poon. Incomplete information in relational temporal databases. In *Proc. Conf. on Very Large Databases*, Vancouver, Canada, August 1992.

[Gol81] B. Goldstein. Constraints on null values in relational databases. In *Proc. of the Int. Conf. on Very Large Databases*, pages 101–110, Cannes, France, September 1981.

[GR95] P. Gardenfors and H. Rott. Belief revision. In D. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume IV: Epistemic and Temporal Reasoning, pages 35–132. Oxford University Press, 1995.

[Gra77] J. Grant. Null values in a relational data base. *Information Processing Letters*, 6(5):156–157, October 1977.

[Gra91] G. Grahne. *The Problem of Incomplete Information in Relational Databases*. Springer LNCS No. 554, 1991.

[Gre69] C. Green. Theorem-proving by resolution as a basis for question answering systems. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 183–205. American Elsevier Publishing Co., 1969.

[GRS91] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[GZ88] G. Gottlob and R. Zicari. Closed world databases opened through null values. In *Proc. Int. Conf. on Very Large Databases*, pages 50–61, Los Angeles, CA, 1988.

[Heg87] S. Hegner. Specification and implementation of programs for updating incomplete information databases. In *Proc. ACM Symp. on Principles of Database Systems*, San Diego, CA, March 1987.

[IL83] T. Imielinski and W. Lipski, Jr. Incomplete information and dependencies in relational databases. In *ACM SIGMOD International Conference on Management of Data*, pages 178–184, 1983.

[IL84] T. Imielinski and W. Lipski, Jr. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

[IL89] T. Imielinski and W. Lipski, Jr. Epilogue to 'Incomplete information in a relational database'. In M. L. Brodie and J. Mylopoulos, editors, *Readings in Artificial Intelligence and Databases*. Springer-Verlag, Berlin, 1989.

[Imi89] T. Imielinski. Incomplete information in logical databases. *IEEE Database Engineering Bulletin - Special Issue on Imprecision in Databases*, 12(2):29–40, June 1989.

[Imi91a] T. Imielinski. Abstraction in query processing. *Journal of the ACM*, 38(1):534–558, 1991.

[Imi91b] T. Imielinski. Incomplete deductive databases. *Annals of Mathematics and Artificial Intelligence*, 3(2-4):259–293, 1991.

[IMV95] T. Imielinski, R. van der Meyden, and K. Vadaparty. Complexity tailored design: A new design methodology for databases with incomplete information. *Journal of Computer and System Sciences*, 51(3):405–432, Dec 1995.

[INV91a] T. Imielinski, S. Naqvi, and K. Vadaparty. Incomplete objects—A data model for design and planning applications. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 288–297, Denver, CO, May 1991.

[INV91b] T. Imielinski, S. Naqvi, and K. Vadaparty. Querying design and planning databases. In *Proc. Int. Conf. on Deductive and Object-Oriented Databases*, Munich, Germany, December 1991.

[JFM92] Y. Jia, Z. Feng, and M. Miller. A multivalued approach to handle nulls in RDB. In *Future Database'92, Proceedings of the Second Far-East Workshop on Future Database Systems*, pages 71–76, Kyoto, Japan, April 1992.

[JL87] J. Jaffar and J-L. Lassez. Constraint logic programming. In *Proc. ACM Symp. Principles of Programming Languages*, pages 111–119, 1987.

[JLP92] A. Jung, L. Libkin, and H. Puhlmann. Decomposition of domains. In Stephen Brookes, Michael Main, Austin Melton, and Michael Mislove, editors, *Proc. of Mathematical Foundations of Programming Semantics. 7th Int. Conf., Pittsburgh, PA, USA, March 25-28, 1991: Proceedings*, volume 598 of *LNCS*, pages 235–258, Berlin, Germany, March 1992. Springer.

[JP95] A. Jung and H. Puhlmann. Types, logic, and semantics for nested databases. In M. Main and S. Brookes, editors, *11th Conf. on Mathematical Foundations of Programming Semantics*, volume 1 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers B.V., 1995.

[Kan90] P. C. Kanellakis. Elements of relational database theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 1071–1156. MIT Press, Cambridge, Mass., 1990.

[Kel86] A. M. Keller. Set-theoretic problems of null completion in relational databases. *Information Processing Letters*, 22(5):261–265, April 1986.

[KKR95] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, 1995.

[KM92] H. Katsuno and A. Mendelzon. On the difference between updating a knowledge base and revising it. In P. Gardenfors, editor, *Belief*

Revision, pages 183–203. Cambridge University Press, Cambridge, 1992.

[Kou94a] M. Koubarakis. Database models for infinite and indefinite temporal information. *Information Systems*, 19(2):141–173, 1994.

[Kou94b] M. Koubarakis. Foundations of indefinite constraint databases. In A. Borning, editor, *Proc. of the 2nd Int. Workshop on the Principles and Practice of Constraint Programming*, Springer LNCS No. 874, pages 266–280, 1994.

[Kou97] M. Koubarakis. The complexity of query evaluation in indefinite temporal constraint databases. *Theoretical Computer Science*, 171(1-2):25–60, 1997.

[Kow74] R. A. Kowalski. Predicate logic as a programming language. In *Proceedings IFIP Congress*, pages 569–574, Amsterdam, 1974. North Holland Publishing Co.

[Kow78] R. A. Kowalski. Logic for data description. In H. Gallaire J. Minker, editor, *Logic and Data Bases*, pages 77–103. Plenum Press, New York, 1978.

[KW84] A. M. Keller and M. W. Wilkins. Approaches for updating databases with incomplete information and nulls. In *Proc. of the Int. Conf. on Data Engineering*, pages 332–340, Los Angeles, CA, April 1984. IEEE Computer Society, IEEE Computer Society Press.

[KW85] A. M. Keller and M. W. Wilkins. On the use of an extended relational model to handle changing incomplete information. *IEEE Transactions on Software Engineering*, 11(7):620–633, July 1985.

[Kwa91] K.L. Kwast. The incomplete database. In *Proc. Int. Joint Conf. on Artificial Intelligence*, pages 897–902, 1991.

[Ler86] N. Lerat. Query processing in incomplete logical databases. In G. Ausiello and P. Atzeni, editors, *Proc. of the Int. Conf. on Database Theory*, pages 260–277, Rome, Italy, September 1986. Springer-Verlag.

[Lev84] H. J. Levesque. The logic of incomplete databases. In M. Brodie, J. Mylopoulos and J. W. Schmidt, editors, *On Conceptual Modeling: Perspectives from Artificial Intelligence Databases and Programming Languages*, pages 165–186. Springer-Verlag, Berlin and New York, 1984.

[Lev92] M. Levene. *The Nested Universal Relation Database Model.* Springer LNCS No. 595, 1992.

[Lev96] A. Y. Levy. Obtaining complete answers from incomplete databases. In *Proc. of the 22nd Int. Conf. on Very Large Databases*, pages 402–412, 1996.

[LG94] L. Libkin and E. Gunter. OR-SML: A functional database programming language with support for disjunctive information. In D. Karagiannis, editor, *Proc. 5th Int. Conf. on Database and Expert Systems Applications (DEXA)*, volume 856, pages 641–650, Athens, Greece, September 1994. Springer-Verlag, Lecture Notes in Computer Science.

[Lib91] L. Libkin. A relational algebra for complex objects based on partial information. In B. Thalheim, editor, *Proc. of the Symp. on Math. Fundamentals of Database Systems (MFDBS)*, volume 495, pages 136–147, Rostock, Germany, May 1991. Springer-Verlag, Lecture Notes in Computer Science.

[Lib94] L. Libkin. *Aspects of partial information in databases.* PhD thesis, Computer and Information Science, University of Pennsylvania, 1994.

[Lib95] L. Libkin. Normalizing incomplete databases. In *Proc. ACM Symp. on Principles of Database Systems*, pages 219–230, 1995.

[Lip79] W. Lipski, Jr. On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems*, 4(3):262–296, September 1979.

[Lip81] W. Lipski, Jr. On databases with incomplete information. *Journal of the ACM*, 28(1):41–70, 1981.

[Lip84] W. Lipski, Jr. On relational algebra with marked nulls. In *Proc. ACM Symp. on Principles of Database Systems*, pages 201–203, Waterloo, Ontario, Canada, April 1984.

[LL86] N. Lerat and W. Lipski, Jr. Nonapplicable nulls. *Theoretical Computer Science*, 46:67–82, 1986.

[LL90] M. Levene and G. Loizou. The nested relation type model: An application of domain theory to databases. *The Computer Journal*, 33:19–30, 1990.

[LL91] M. Levene and G. Loizou. Correction to null values in nested relational databases by M. A. Roth, H. F. Korth, and A. Silberschatz. *Acta Informatica*, 28:603–605, 1991.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, second edition, 1987.

[LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT Press, Cambridge, MA, 1992.

[LT85] J. W. Lloyd and R. Topor. A basis for deductive database systems. *Journal of Logic Programming*, 2:93–109, 1985.

[LW96] L. Libkin and L. Wong. Semantic representations and query languages for or-sets. *Journal of Computer and System Sciences*, 52(1):125–142, February 1996.

[Mai80] D. Maier. Discarding the universal instance assumption: Preliminary results. In *XP1 Workshop on Relational Database Theory*. SUNY at Stony Brook, NY, June-July 1980.

[Mai83] D. Maier. *The Theory of Relational Databases*, chapter 12: Null Values, Partial Information, and Database Semantics. Computer Science Press, Rockville, MD, 1983.

[Men64] E. Mendelson. *Introduction to Mathematical Logic*. D. van Nostrand Co., New York, 1964.

[Mey93] R. van der Meyden. Recursively indefinite databases. *Theoretical Computer Science*, 116:151–194, 1993.

[Mey97] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *Journal of Computer and System Sciences*, 54(1):113–135, Feb 1997.

[Min82] J. Minker. On indefinite databases and the closed world assumption. In *6th Conference on Automated Deduction*, pages 292–308. Springer LNCS No. 138, 1982.

[Moo85] R. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–91, 1985.

[Mot89] A. Motro. Integrity = validity + completeness. *ACM Transactions on Database Systems*, 14(4):480–502, December 1989.

[MP84] J. Minker and D. Perlis. Applications of protected circumscription. In *Proc. of the 7th Conference on Automated Deduction*, pages 414–425. Springer, 1984.

[MP85] J. Minker and D. Perlis. Computing protected circumscription. *Journal of Logic Programming*, 2(4):235–249, December 1985.

[MS96] A. Motro and P. Smets, editors. *Uncertainty Management in Information Systems*. Kluwer Academic Publishers, Boston, 1996.

[MT91] V. W. Marek and M. Truszczynski. Autoepistemic logic. *Journal of the ACM*, 38(3):588–619, 1991.

[MvdM92] L. T. McCarty and R. van der Meyden. Reasoning about indefinite actions. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 59–70. Morgan Kaufmann, 1992.

[NG78] J. M. Nicholas and H. Gallaire. Data base: theory vs. interpretation. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 33–54. Plenum Press, New York, 1978.

[Oho90a] A. Ohori. Orderings and types in databases. In F. Bancilhon and P. Buneman, editors, *Advances in Database Programming Languages*, pages 97–116. ACM Press, 1990.

[Oho90b] A. Ohori. Semantics of types for database objects. *Theoretical Computer Science*, 76(1):53–91, 31 October 1990.

[Osb81] S. Osborn. Insertions in a multi-relation database with nulls. In *Proc. of COMPSAC81: IEEE Computer Society's Fifth Int. Computer Software and applications Conference*, pages 75–80, Chicago, IL, November 1981.

[Ost87] P. Ostermann. Modal logic and incomplete information. In *Proc. First Symposium on Mathematical Fundamentals of Database Systems*, pages 181–196, Dresden, GDR, January 1987.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[PP90] T. Przymusinski and H. Przymusinska. Semantic issues in deductive databases and logic programs. In R. Banerji, editor, *Formal Techniques in Artificial Intelligence. A Source-book*, pages 321–367. North Holland, 1990.

[Prz89] T.C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufman, 1989.

[Prz91] T. Przymusinski. Stable semantics for disjunctive programs. *New Generation Computing*, 9:401–424, 1991.

[Prz93] T. Przymusinski. Logic programming and non-monotonic reasoning. *Journal of Logic Programming*, 17:91–94, 1993. Special Issue on Non-Monotonic Reasoning.

[Prz95] T. Przymusinski. Static semantics for normal and disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 14:323–357, 1995.

[PY97] C. H. Papadimitriou and M. Yannakakis. On the complexity of database queries. In *Proc. ACM Symp. on Principles of Database Systems*, 1997.

[Rei78a] R. Reiter. Deductive question answering on relational databases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 149–178. Plenum, New York, 1978.

[Rei78b] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum, New York, 1978.

[Rei84] R. Reiter. Towards a logical reconstruction of relational database theory. In M. Brodie, J. Mylopoulos and J. W. Schmidt, editors, *On Conceptual Modeling: Perspectives from Artificial Intelligence Databases and Programming Languages*, pages 191–238. Springer-Verlag, Berlin and New York, 1984.

[Rei86] R. Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *Journal of the ACM*, 33(2):349–370, 1986.

[Rei88] R. Reiter. On integrity constraints. In M. Y. Vardi, editor, *Proc. Conf. Theoretical Aspects of Reasoning about Knowledge*, pages 97–112, 1988.

[Rev93] P. Z. Revesz. On the semantics of theory change: Arbitration between new and old information. In *Proc. ACM Symp. on Principles of Database Systems*, pages 71–82, 1993.

[RKS89] M. A. Roth, H. F. Korth, and A. Silberschatz. Null values in nested databases. *Acta Informatica*, 26:615–642, 1989.

[RKS91] M. A. Roth, H. F. Korth, and A. Silberschatz. Addendum to null values in nested relational databases. *Acta Informatica*, 28:607–610, 1991.

[RLM89] A. Rajasekar, J. Lobo, and J. Minker. Weak generalized closed world assumption. *Journal of Automated Reasoning*, 5:293–307, 1989.

[Roy91] V. Royer. The semantics of incomplete databases as an expression of preferences. *Theoretical Computer Science*, 78(1):113–136, January 1991.

[RT88] K. A. Ross and R. W. Topor. Inferring negative information from disjunctive databases. *Journal of Automated Reasoning*, 4(2):397–424, 1988.

[Sak89] C. Sakama. Possible model semantics for disjunctive databases. In *Proc. of the Int. Conf. on Deductive and Object-Oriented Databases (DOOD'90)*, Kyoto, Japan, December 1989.

[Sci80] E. Sciore. *The Universal Instance and Database Design*. PhD thesis, Princeton University, Princeton, NJ, 1980.

[She88] J. Shepherdson. Negation in logic programming. In J. Minker, editor, *Deductive Databases and Logic Programming*, pages 19–88. Morgan Kaufmann, Los Altos, CA, 1988.

[SRR94] D. Srivastava, R. Ramakrishnan, and P. Z. Revesz. Constraint objects. In *Proc. of the 2nd Int. Workshop on Principles and Practice of Constraint Programming*, Springer LNCS No. 874, pages 218–228, 1994.

[Sto86] M. Stonebraker, editor. *The INGRES Papers: Anatomy of a Relational Database System*. Addison-Wesley, Reading, Mass., 1986.

[Sub90] V. S. Subrahmanian. Paraconsistent disjunctive deductive databases. In *Proc. of the 20th Int. Symp. on Multiple-Valued Logic*, pages 339–346, Charlotte, NC, May 1990.

[TCG+93] A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings, Redwood City, Cal., 1993.

[Ull88] J.D. Ullman. *Principles of Database and Knowledge Base Systems, volume I*. Computer Science Press, 1988.

[Var82] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 137–146, 1982.

[Var86a] M. Y. Vardi. On the integrity of databases with incomplete information. In *Proc. ACM Symposium on Principles of Databases*, pages 252–266, 1986.

[Var86b] M. Y. Vardi. Querying logical databases. *Journal of Computer and System Sciences*, 33:142–160, 1986.

[Var95] M. Y. Vardi. On the complexity of bounded-variable queries. In *Proc. ACM Symp. on Principles of Database Systems*, pages 266–276, May 1995.

[Vas79] Y. Vassiliou. Null values in database management: a denotational semantics approach. In *Proc. of the 1979 ACM SIGMOD Int. Conf. on Management of Data*, pages 162–169, New York, May 1979. ACM Press.

[Vas80] Y. Vassiliou. Functional dependencies and incomplete information. In *Int. Conf. on Very Large Databases*, pages 260–269, October 1980.

[Win90] M. Winslett. *Updating Logical Databases*. Cambridge University Press, Cambridge, 1990.

[YFM92] Y. Yia, Z. Feng, and M. Miller. A multivalued approach to handle nulls in RDB. In *Proc. 2nd Far-East Workshop on Future Database Systems*, Kyoto, Japan, April 1992.

[YH85] A. Yahya and L. J. Henschen. Deduction in non-Horn databases. *Journal of Automated Reasoning*, 1(2):141–160, 1985.

[Yue91] K. Yue. A more general model for handling missing information in relational databases using A 3-valued logic. *ACM SIGMOD Record*, 20(3):43–49, September 1991.

[Zan84] C. Zaniolo. Database relations with null values. *Journal of Computer and System Sciences*, 28:142–166, 1984.

[Zic90] R. Zicari. Incomplete information in object-oriented databases. *ACM SIGMOD Record*, 19(3):5–16, September 1990.

[ZP96] E. Zimanyi and A. Pirotte. Imperfect knowledge in relational databases. In A. Motro and P. Smets, editors, *Uncertainty Management in Information Systems*. Kluwer Academic Publishers, Boston, 1996.