Incomplete Data: What Went Wrong, and How to Fix It

Leonid Libkin
School of Informatics
University of Edinburgh
libkin@inf.ed.ac.uk

ABSTRACT

Incomplete data is ubiquitous: the more data we accumulate and the more widespread tools for integrating and exchanging data become, the more instances of incompleteness we have. And yet the subject is poorly handled by both practice and theory. Many queries for which students get full marks in their undergraduate courses will not work correctly in the presence of incomplete data, but these ways of evaluating queries are cast in stone – SQL standard. We have many theoretical results on handling incomplete data but they are, by and large, about showing high complexity bounds, and thus are often dismissed by practitioners. Even worse, we have a basic theoretical notion of what it means to answer queries over incomplete data, and yet this is not at all what practical systems do.

Is there a way out of this predicament? Can we have a theory of incompleteness that will appeal to theoreticians and practitioners alike, by explaining incompleteness and being at the same time implementable and useful for applications? After giving a critique of both the practice and the theory of handling incompleteness in databases, the paper outlines a possible way out of this crisis. The key idea is to combine three hitherto used approaches to incompleteness: one based on certain answers and representation systems, one based on viewing incomplete databases as logical theories, and one based on orderings expressing relative value of information.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*Data Models*; H.2.1 [**Database Management**]: Languages—*Query Languages*; H.2.4 [**Database Management**]: Systems—*Query Processing*

PODS'14, June 22–27, 2014, Snowbird, UT, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2375-8/14/06 ...$15.00. http://dx.doi.org/10.1145/2594538.2594551
.

General Terms

Theory, Languages, Algorithms

Keywords

Incomplete information, query evaluation

1. INTRODUCTION

The need to handle incomplete data was recognized early in the development of relational database systems: already in the 1970s, Codd developed the basis of what would become null-related features of commercial DBMSs [21, 22]. His proposal for a single one-size-fits-all null value, its propagation through arithmetic and Boolean operations, and the use of the three-valued logic for computing with nulls was largely reflected in the SQL standard. It has, however, quickly become apparent that the adopted design of null-related features has a number of deficiencies, and it has become one of the most criticized aspects of SQL design [24, 26].

To illustrate one of the best known points of such criticism, consider a database of orders and payments, with relations `Order(o_id,product)` and `Pay(p_id,order,amount)`: the first gives order ids and products they are for, the other indicates that a payment with a given id was made for an order. We want to check if there are unpaid orders. A student who has taken a basic database course will immediately produce

```
SELECT o_id FROM Order
WHERE o_id NOT IN (SELECT order FROM Pay)
```

expecting to get full marks. But now take `Order` $= \{(\mathrm{oid1}, \mathrm{pr1}), (\mathrm{oid2}, \mathrm{pr2})\}$, and `Pay` $= \{(\mathrm{pid1}, \bot, 100)\}$, where $\bot$ indicates null value. We know that at least one order has not been paid for, and yet the above query happily returns the empty set, indicating that no customers need to be chased for their payments! The problem easily confounds SQL programmers: consider a simple query $R - S$, where R and S are single-attribute relations, written as `SELECT R.A FROM R WHERE R.A NOT IN (SELECT S.A FROM S)`. It will produce the empty set if S contains just a null value, no matter what R contains. This goes against our intuition, of course: we know that if $|R| > |S|$, then $R - S$ cannot possibly be empty, but SQL tells us that it is. As [26] nicely put it, *"those SQL features are ... fundamentally at odds with the way the world behaves"*; a more damning assertion *"you can never trust the answers you get from a database with nulls"* is found in [24].

How did we get there? The approach to evaluating queries with nulls dates back to Codd's paper [21] from 1975, in which the 3-valued logic approach is advocated. Problems with it were caught early [37]: a simple query

```
SELECT p_id
FROM Pay
WHERE order = "oid1"  OR  order <> "oid1"
```

when evaluated on the database shown above, produces the empty table, and yet intuitively we expected the answer to be 'pid1'. Indeed, no matter what non-null value we replace the null with, this is what the query will produce.

The idea of answering queries consistently with every possible interpretation of nulls, first proposed in [37] as a way of fixing some problems with Codd's 3-valued approach, led to the notion of *certain answers*, now the standard way of answering queries over incomplete databases. It was first properly defined by [54]. The definition relies on the notion of a *semantics* of an incomplete database D, denoted by $[\![D]\!]$, which is the set of all complete databases D' that D can represent. For instance, such databases can be obtained by replacing nulls by values (but this is not the only possibility). Then, given a relational query Q and an incomplete database D, certain answers were defined as

$$\mathsf{certain}(Q, D) \;=\; \bigcap \{Q(D') \mid D' \in [\![D]\!]\}, \qquad (1)$$

i.e., they consist of tuples that belong to the answer no matter how the missing information is interpreted.

In the theory community, certain answers have become *the* way for answering queries over incomplete databases, used across a variety of applications such as query answering using views [1, 39], data integration [43], data exchange [7], inconsistency management [15], and data cleaning [30].

However, this more disciplined approach, compared to SQL's 3-valued logic, does not come for free. Let Q be a Boolean (i.e., true/false) query. Then $\mathsf{certain}(Q, D)$ is true iff Q is true in every $D' \in [\![D]\!]$. Thus computing certain answers becomes a form of *validity* (i.e., checking if a sentence is true in all structures). In fact, when Q comes from relational calculus, then under the open-world-semantics (to be defined later), finding certain answers is exactly the validity problem. However, this is an undecidable problem, as was shown by Church and Turing back in the 1930s. Actually, the problem we look at is slightly different – we are only concerned about finite structures – but that does not make it easier. A detailed study of the complexity of finding certain answers was initiated by [3] which showed hardness results, coNP-hard and up. In fact, high complexity bounds are widespread in applications of incompleteness as well, with classes such as coNP, Π_2^p, Pspace, coNEXP, and so on being regularly mentioned, even for *data* complexity.

So, where does this bring us? We can summarize the state of affairs roughly as follows:

Practice:
- sacrifice correctness for efficiency;
- the same query evaluation engine for complete and incomplete data;

Theory:
- correctness at the expense of efficiency;
- new semantics for query answering in the presence of incompleteness.

The picture looks quite bleak: it is almost 40 years since nulls were introduced, and yet the practice has taught generations to live with incorrect answers, while the theory is not really addressing the right problems. What can we do?

First, we have to recognize that we live in the real world, and no database vendors will change their products if we offer them radical solutions, like completely new query evaluation algorithms. Indeed, it took them many years to make DBMSs as efficient as they are today, and significant changes in basic query processing algorithms will result in years worth of work to adjust other elements of their products. We can, perhaps, suggest small and easily implementable changes. And we definitely can suggest new algorithms for specialized products.

Second, we must develop a theory applicable to a variety of models: relational, XML, graph data, with different types of incomplete information. But we also need a good testbed for such a general theory. Nulls seen in earlier examples correspond to SQL's view of missing information in standalone databases. But we can have incompleteness due to a multitude of other reasons, in particular, data interoperability. Incompleteness inevitably arises when we move data between different applications, such as in data integration and exchange scenarios [5, 7, 29, 43]. For example, suppose that from the `Order` relation, we want to build a database of customers and their preferences. Such a transformation is usually specified by rules known as schema mappings [7], for instance,

$$\texttt{Order}(i, p) \rightarrow \texttt{Cust}(x), \texttt{Pref}(x, p)$$

saying that if an order was placed for a product p, then a customer x must exist who placed that order, and that customer x prefers product p. From the tuple `Order`(oid1,pr1), this rule will generate tuples `Cust`$(\bot)$ and `Pref`$(\bot$,pr1), and from the tuple `Order`(oid2,pr2) it will generate `Cust`$(\bot')$ and `Pref`$(\bot'$,pr2). Note that it is important for us to remember that, while the values $\bot$ and $\bot'$ are not yet known, when $\bot$ is replaced by some value c in Cust($\bot$), it must be replaced by the *same* value c in Pref($\bot$,pr1), and likewise for $\bot'$. On the other hand, $\bot$ and $\bot'$ may be replaced by the same, or by different constants – there are no restrictions.

What this example tells us is that we must have a mechanism for saying that some nulls should always be replaced by the same constant. Such nulls are known as *naïve*, or *marked* nulls [2, 40]. They are the most common model of nulls used in integration/exchange tasks [7, 29, 43] and in fact have been implemented as part of schema mapping and data exchange tools [38, 55].

Thus, while the approach to incompleteness we are about to present does not assume any particular data model, we shall be using, as the main illustration, the model of naïve nulls in relations (of course SQL's nulls are just a special case of it). In this model the standard interpretation of nulls is that a value is missing. This is not the only possibility: other nulls, such as 'non-applicable' or 'no-information' exist as well [44, 67]. However, all the results that we show here apply regardless of the nature of nulls: all that we need is a definition of the semantics of incomplete databases for results to work.

Plan of the paper Our goal is to make an attempt at building applicable theory of incompleteness. We start by recalling the basics of existing relational theory of incomplete information in Section 2. In Section 3 we discuss a number of serious shortcomings of this theory.

In Section 4 we explain the basics of a different approach to incompleteness, based on a duality between objects and queries. Combining the two, we present a simple model of incomplete objects (not just relational databases) that lets us define the notion of certainty in a principled way. We do it in Section 5, and show that there are actually two different notions of certainty: one represents it as an object, and the other as the knowledge we possess about that object.

We then use the new notions of certainty to apply them to query answers and extract what should rightly be called certain answers to queries. This is done in Section 6, which shows that sometimes it is actually very easy to compute certain answers using existing query evaluation technology. The key is the right notion of the semantics, of both input databases and query answers, and the right representational mechanism for query answers. Section 7 outlines directions for further work.

2. RELATIONAL INCOMPLETENESS

We now present formal definitions for some of the basic notions related to incomplete information in relational databases, see [2, 34, 40, 66]. But first, we briefly recall the main languages we deal with here, cf. [2]. The basic language will be *relational algebra*, on the procedural side, and *first-order logic* (FO), or relational calculus, on the declarative side. The selection-projection-join-union fragment of relational algebra is also referred to as the *positive* relational algebra (the difference operator is removed). Logically, it corresponds to the $\exists, \wedge, \vee$-fragment of FO, also known as existential positive formulae. In terms of its expressiveness, it is exactly the same as *unions of conjunctive queries*, denoted by UCQ. Recall that conjunctive queries are select-project-join, or $\exists, \wedge$-queries; queries in UCQ are their unions.

Incomplete databases and their semantics

We assume that databases are populated by two types of elements: constants (such as numbers, strings, etc.) and nulls. The set of constants is denoted by Const and the set of nulls by Null. These are countably infinite sets. Nulls will be denoted by $\bot$, sometimes with sub- or superscripts.

A relational schema is a set of relation names with associated arities. An incomplete relational instance D assigns to each k-ary relation symbol S from the schema a k-ary relation over Const $\cup$ Null, i.e., a finite subset of $(\mathsf{Const} \cup \mathsf{Null})^k$. Such incomplete relational instances are referred to as *naïve* databases [2, 40]; note that a null $\bot \in$ Null can appear multiple times. If each null $\bot \in$ Null appears at most once, we speak of *Codd* databases; these model SQL's nulls. If we talk about single relations, it is common to refer to them as naïve tables and Codd tables.

We write Const(D) and Null(D) for the sets of constants and nulls that occur in a database D. The *active domain* of D is adom(D) = Const(D) $\cup$ Null(D). A *complete* database D has no nulls, i.e., adom(D) $\subseteq$ Const.

Below, R is a naïve table and S is a Codd table:

R:

$\bot$	1	$\bot'$
2	$\bot'$	$\bot$

S:

$\bot_1$	1	$\bot_2$
2	$\bot_3$	$\bot_4$

with Const(R) = Const(S) = $\{1, 2\}$ and Null(R) = $\{\bot, \bot'\}$ and Null(S) = $\{\bot_1, \bot_2, \bot_3, \bot_4\}$.

Each incomplete database can represent many possible complete databases. The exact set of complete databases it represents is the *semantics* of an incomplete database. The semantics is by no means unique, but in this paper we concentrate on the two most common ones, based on *open-world* and *close-world* assumptions [40, 58], usually abbreviated as OWA and CWA. The key notion for both is a *valuation* of nulls, which is a mapping v : Null(D) $\to$ Const. This mapping associates a constant value with each null. It naturally extends to databases: $v(D)$ is simply the result of replacing each null $\bot \in$ adom(D) by $v(\bot)$.

With that, we define CWA and OWA semantics as follows:

$$\begin{array}{lcl} [\![D]\!]_{\text{CWA}} & = & \{D' \mid D' = v(D),\ v \text{ is a valuation}\} \\ [\![D]\!]_{\text{OWA}} & = & \{D' \mid D' \supseteq v(D),\ v \text{ is a valuation}\} \end{array}$$

Under CWA, we believe that an incomplete database represents information fully, except some missing values. Thus, databases represented by it are obtained by substituting values for nulls. Under OWA, the database is open to adding new facts: thus, after substituting values for nulls, one can add new tuples.

For instance, relation R_1 below belongs to both $[\![R]\!]_{\text{CWA}}$ and $[\![R]\!]_{\text{OWA}}$, for R depicted above (as it is obtained by valuation $\bot \mapsto 3,\ \bot' \mapsto 4$), and relation R_2 is in $[\![R]\!]_{\text{OWA}}$, as it also adds the tuple $(5, 6, 7)$:

R_1:

3	1	4
2	4	3

R_2:

3	1	4
2	4	3
5	6	7

A more expressive representation mechanism for incomplete information is that of *conditional tables*. Such a table is of the form

$$D \;=\; \left(\begin{array}{|c|c|} \hline & \text{condition} \\ \hline t_1 & c_1 \\ \hline \cdots & \cdots \\ \hline t_n & c_n \\ \hline \end{array} \;,\; c \right)$$

where $t_1, \ldots, t_n$ are tuples and $c, c_1, \ldots, c_n$ are *conditions*: Boolean combinations of statements $x = y$, where $x, y \in \mathsf{Const} \cup \mathsf{Null}$. Note that conditions may use nulls not present in the tuples. Conditional tables are usually viewed under the closed-world semantics $[\![D]\!]_{\text{CWA}}$ which consists of databases $\{v(t_i) \mid v(c_i) = \text{true},\ i \leq n\}$, where v is a valuation so that $v(c)$ is true. For example, consider a conditional table

$$D \;=\; \left(\begin{array}{|c|c|} \hline & \text{condition} \\ \hline 1 & \bot = 1 \\ \hline 0 & \bot = 0 \\ \hline \end{array} \;,\; (\bot = 0) \vee (\bot = 1) \right)$$

The only valuations satisfying $(\bot = 0) \vee (\bot = 1)$ are $\bot \mapsto 0$ and $\bot \mapsto 1$. Hence $[\![C]\!]_{\text{CWA}} = \{\{0\},\ \{1\}\}$; conditional tables thus can encode disjunctions: C says that either 0 or 1 is in the database.

Query answering

Fix a semantics $[\![\,]\!]$ of incompleteness, and assume we are given a query Q and an incomplete database D. Of course we know how to evaluate Q on complete databases. So the key object for us to work with is

$$Q([\![D]\!]) \;=\; \{Q(D') \mid D' \in [\![D]\!]\}$$

of query answers on all databases that are possibly represented by D. If we have an incomplete database that represents this set, then we have our query answer. That is, if there is a table A (for *answer*) such that

$$[\![A]\!] \;=\; Q([\![D]\!]), \tag{2}$$

then we declare A to be the answer to Q on D. Indeed, we get a single table that captures exactly the space of all possible query answers. If this happens for all incomplete database from some class $\mathcal{K}$ (Codd/naïve/conditional tables, or others) and for all queries Q from a language $\mathcal{L}$, then we say that $\mathcal{K}$ forms a *strong representation system* for $\mathcal{L}$ under $[\![\,]\!]$. Strong representation systems, as the name suggest, are quite strong, and thus are hard to come by. The best known example is that of conditional tables for full relational algebra (equivalently, first-order logic) under $[\![\,]\!]_{\text{CWA}}$, see [40]. To give an example, let us revisit the query $R - S$. If our database D has $R = \{1, 2\}$ and $S = \{\bot\}$, then $Q([\![D]\!]_{\text{CWA}}) = \{\{1, 2\}, \{1\}, \{2\}\}$, depending on whether the null $\bot$ is instantiated into 1, or 2, or another constant. This can be represented by a conditional table

	condition
1	$\bot' = 1 \vee \bot' = 2$
2	$\bot' \neq 1$

Indeed, going over possible values of $\bot'$, one can see that it generates exactly $Q([\![D]\!]_{\text{CWA}})$. One problem with such an answer is that it is hardly meaningful to humans, and one probably would not be happy getting this answer from a DBMS.

Since (2) is too strong a condition, one tries to replace it by

$$[\![A]\!] \;\sim\; Q([\![D]\!]) \tag{3}$$

where $\sim$ is some equivalence relation. This idea led to the notion of a *weak representation system* based on the following equivalence. For two sets of instances, $\mathcal{I}_1$ and $\mathcal{I}_2$, and a query language $\mathcal{L}$, we let $\mathcal{I}_1 \sim_{\mathcal{L}} \mathcal{I}_2$ if $\bigcap\{q(D') \mid D' \in \mathcal{I}_1\} = \bigcap\{q(D') \mid D' \in \mathcal{I}_2\}$ for each q in $\mathcal{L}$. If (3) holds for $\sim_{\mathcal{L}}$ over a class $\mathcal{K}$ of instances for each query $Q \in \mathcal{L}$, we say that $\mathcal{K}$ forms a weak representation system for $\mathcal{L}$ under semantics $[\![\,]\!]$. The best known examples are, under both $[\![\,]\!]_{\text{OWA}}$ and $[\![\,]\!]_{\text{CWA}}$:

- Codd tables for selection/projection queries; and
- Naïve tables for UCQs (positive relational algebra).

The key reason weak representation systems are of interest is that they let us compute certain answers. Given an instance D, let D^{cmpl} be the complete part of it, i.e., all the tuples in D without nulls. Then, if we have a weak representation system, it follows that $A^{\text{cmpl}} = \mathsf{certain}(Q, D)$. Thus, certain answers are obtained by keeping the complete portion of the answer given by (3).

What makes the connection particularly attractive is that sometimes A is just $Q(D)$, i.e., one *naïvely* evaluates Q on D as if nulls were the usual values. In such a case, when A in (3) equals $Q(D)$, we say that *naive evaluation works for Q*. It then follows that

$$Q(D)^{\text{cmpl}} \;=\mathsf{certain}(Q, D). \tag{4}$$

It is known that naïve evaluation works for UCQs under both open and closed world semantics [40]. Moreover, under OWA, the result is optimal for FO: if we have a Boolean (yes/no) FO query and naïve evaluation works for it, then it is equivalent to a UCQ, i.e., a positive relational algebra query [51]. To see how naïve evaluation fails for non-positive queries, consider the query $\pi_A(R - S)$ where $R = \{(1, \bot)\}$ and $S = \{(1, \bot')\}$ are relations over attributes A, B. Then naïve evaluation computes $\{1\}$, while the certain answer is $\emptyset$.

If naïve evaluation works, i.e., (4) holds, computing certain answers can be done by a straightforward query evaluation following by an extra selection operation, throwing out tuples with nulls (or simply adding `IS NOT NULL` conditions in the `WHERE` clause of the original query). Thus, we do not need to invent new evaluation techniques.

As for the complexity of computing certain answers, for full relational algebra (first-order logic) it is undecidable under $[\![\,]\!]_{\text{OWA}}$ and coNP-complete under $[\![\,]\!]_{\text{CWA}}$, for data complexity (that is, for a fixed query, when only database is the input), see [3, 33]. This makes it prohibitively expensive under CWA, and plain impossible under OWA, but the good news is that due to (4), the complexity is very low ($\text{AC}^0 \subsetneq \text{DLogSpace}$) for positive relational algebra queries.

It is a general phenomenon that by going away from positive queries, one loses tractability of finding certain answers, demonstrated for many problems related to handling incompleteness in databases [7, 13, 15, 17, 18, 29, 43, 53, 62, 66]. There are some classes extending UCQs for which certain answers can be computed tractably – for instance, Boolean combinations of conjunctive queries [33] – but the algorithms, despite having polynomial time bounds, are too complicated to be efficiently implemented on top of existing DBMSs.

3. CRITIQUE OF THEORY

There has been plenty of criticism of practical approaches to incomplete information, in particular SQL's treatment of nulls, see, e.g., [24, 25, 26], but the theoretical approaches have so far been spared. We put an end to it now. In fact much of theoretical research on incomplete information took the notions of *strong and weak representation systems* and *certain answers* as sacrosanct but we shall argue that their untouchable status needs to be re-examined.

Semantics of query answers. Let us look at the seemingly uncontroversial (2) saying that if we are lucky enough to get A satisfying $[\![A]\!] = Q([\![D]\!])$, then A should be viewed as the answer to Q on D. At the first glance it looks like a reasonable condition, but nonetheless there is one assumption built into it that is not unassailable. Note that (2) requires that both the input database D, and the answer A, be interpreted under the *same* semantics $[\![\,]\!]$. However, a priori, there is no reason for it. Why, for instance, should the answer to a query be interpreted under CWA if this is the semantics of the input?

Why intersection? The equivalence $\sim_{\mathcal{L}}$ used in the definition of weak representation systems looks quite ad hoc. Actually it is: it was defined that way to ensure compositionality, but its essence is really going from the very strong requirement (2) to a weaker one that only certain answers need to be produced. And certain answers are defined as the intersection of all possible answers. Again, at first this looks very reasonable: we want tuples that will be in the answer no matter how nulls are interpreted. But a closer examination reveals some problems. To start with, there are models other than relational. What can one do, for instance, for XML queries returning documents? (A side remark: much of the work on incompleteness in XML has been restricted to XML-to-relational queries, for this very reason [4, 9, 13, 33].) But even more importantly, how do we know that we do not lose important information by taking intersection and removing information from the answer?

Are certain answers certain? Actually, the standard intersection-based certain answers need not be. Intersection takes some tuples away from potential answers. At first the intuition appears to be fine: removing tuples from what is certain, we seem to retain only information we are certain about. However, removing tuples amounts to removing *data*, not information. In fact, the process can actually *add* information: for instance, under CWA, by removing a tuple we gain information that it is not in the answer. Hence, certain answers defined by (1) cannot be called certain in all scenarios.

Semantics and informativeness. Above, we alluded to the possibility of comparing incomplete databases in terms of their informativeness. This is a line of work that was pursued in the 1990s, rather independently of the rest of the work on incompleteness [16, 49, 57, 64]. The idea was to define orderings stating that one database has more information than another, albeit for primitive models, such as Codd tables. Having an ordering describing informativeness could be important for deciding what the proper semantics of query answers is, bringing us back to our first point of discussion. Indeed, it is expected that one should get more informative answers from more informative databases. However, there was no real attempt to tie the ordering-based approach with the basics such as representation systems and certain answers, and this needs to be done.

Are objects sufficiently expressive to be query answers? We are used to queries returning database objects – tables, XML documents, graphs. But are these sufficiently expressive to describe answers on incomplete databases? Specifically, are these sufficiently expressive to represent sets $Q([\![D]\!])$? Such sets may well be infinite, and describing them may require a more complex representation mechanism than simple database objects: an example of that was already seen when we looked at conditional tables. But is it always possible – and necessary – to have representations that look like database relations, while they are not?

Can high complexity bounds be avoided? Too much work has been done on showing high complexity bounds. A typical picture looks like this: a class of queries, often a fragment of positive relational algebra, can be evaluated efficiently; beyond that, intractability or even undecidability of data complexity is shown. Such results, while occasionally requiring nontrivial machinery, are becoming completely standard – but is this really the direction the field should be going in? And how much of this owes to the rigid setting (basic semantics, certain answers) that one is unwilling to tweak and experiment with? In fact, very often high complexity is shown in the setting where the semantics of input databases is the same as the semantics of query answers. So perhaps there is another reason to reconsider that assumption.

Thus, despite an extensive literature and cast-in-stone notions of representation systems and certain answers, theoreticians do not actually know that much about handling incomplete information in databases. There are very basic questions that are still lacking adequate answers; among them:

- What is the semantics of query answers? When can/should it be the same as the semantics of input databases?
- Is taking intersection the only way to define certain answers?
- What does it mean to have a more informative data set?
- How do informativeness and semantics relate?
- How can we represent answers to queries over incomplete databases?
- When can we rely on existing query evaluation algorithms to produce meaningful answers?

It may seem that neither theory nor practice has good answers to a persistent and ubiquitous problem of handling incomplete information. But perhaps we can view this positively rather than negatively: this simply means that we are back at square one, and an effort must be made to develop a proper theory and to apply it. Both sides must show some flexibility – in tweaking both definitions and products – but first questions posed above (and many others) need to be answered. This paper does not claim to provide all such answers, far from it. But we shall at least attempt to outline an approach: one has to start somewhere, after all. Our idea is to bind together three directions of work on incomplete information:

1. the standard database approach based on representation systems and certain answers;
2. the approach from the knowledge representation community, based on viewing databases as logical theories, pioneered by Reiter [58, 60] in the 1980s; and
3. the approach based on the ideas from programming semantics that used orderings to describe information content, proposed in the 1990s [16, 49].

4. DUALITY: INCOMPLETE DATA AS QUERIES

We now describe an alternative way of looking at incomplete databases that dates back to [58, 60]. It proved to be more popular with the knowledge representation community than with the mainstream database community. More importantly for us, it developed the idea of duality between queries and databases (first noticed in [19]) for incompletely specified databases.

We start with an example. Consider an incomplete relation $R = \{(1, \bot), (\bot, 2)\}$. It can be viewed as a tableau of a Boolean conjunctive query $Q_R = \exists x \ R(1,x) \wedge R(x,2)$. Complete databases satisfying this query are precisely the databases in the semantics of R under OWA. If we let $\mathsf{Mod}_{\mathsf{C}}(\varphi)$ stand for all the models of a formula φ among complete databases, then our observation can be formulated as

$$\mathsf{Mod}_{\mathsf{C}}(Q_R) \ = \ [\![R]\!]_{\mathrm{OWA}} \,. \tag{5}$$

This tells us that the semantics of an incomplete database can be defined by a logical formula. This can be extended for other semantics: for instance, the formula

$$Q_R^{\mathrm{CWA}} = \exists x \left(\begin{array}{l} \qquad R(1,x) \wedge R(x,2) \\ \wedge \ \forall y,z \ (R(y,z) \rightarrow \left(\begin{array}{rc} & y = 1 \wedge z = x \\ \vee & y = x \wedge z = 2 \end{array} \right) \end{array} \right)$$

has the property that $\mathsf{Mod}_{\mathsf{C}}(Q_R^{\mathrm{CWA}}) = [\![R]\!]_{\mathrm{CWA}}$. In general, the approach of [58, 60] was to view a database as a logical theory, i.e., a collection Φ of formulae. A finite $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ can of course be viewed as a single formula $\varphi_1 \wedge \ldots \wedge \varphi_n$.

What is the advantage of viewing incomplete databases as logical theories? An immediate benefit is that we can cast the query answering problem as logical implication, or, closer to the database language, as query containment.

Indeed, suppose we have an database D given as a theory Φ so that $\mathsf{Mod}_{\mathsf{C}}(\Phi) = [\![D]\!]$. Take a Boolean query Q. For it to be true in every database in $[\![D]\!]$ it has to be true in every model of Φ; thus, Q is true with certainty iff it is *logically implied* by Φ, i.e., $\Phi \models Q$. If Φ happens to be a single query Q', as in our examples above, this amounts to checking implication $Q' \models Q$, or, as database literature prefers to call it, *containment* of Q' in Q. Thus, finding certain answers is a special case of logical implication or query containment.

The connection gives us further insights. Suppose we have an incomplete database D, a Boolean conjunctive query Q, and we would like to know whether the certain answer to Q is true on D. As in (5), we have a Boolean conjunctive query Q_D so that $\mathsf{Mod}_{\mathsf{C}}(Q_D) = [\![D]\!]_{\mathrm{OWA}}$. Thus, under OWA, $\mathsf{certain}(Q, D)$ is true iff Q_D is contained in Q. By a well known fact about conjunctive queries, this happens if and only if the tableau of Q_D satisfies Q – but the tableau of Q_D is D itself. Hence, the certain answer is true iff $D \models Q$. Thus, viewing incomplete databases as formulae, we can use known results on containment to find cases when naïve evaluation works.

One may notice that we used conjunctive queries together with the OWA semantics, which is described as the models of conjunctive queries. Is this a coincidence? Is it possible, for instance, to use naïve evaluation for a larger class of queries under CWA, since the formula describing $[\![\,]\!]_{\text{CWA}}$ uses features beyond those of conjunctive queries? We shall see later that the answer is positive.

5. INCOMPLETENESS AND CERTAINTY: A NEW LOOK

With this dual look at incomplete information – as formulae and as structures – we now start addressing questions posed at the end of Section 3. We want to deal with them in a setting that is independent of a particular data model – and yet applicable to all of them. In other words, we do not want to provide definitions that will apply specifically to relational databases, or to XML documents, or to graph databases. What we want instead is a *minimalistic* approach that will use only the key concepts of incompleteness. In fact, it is easier to work in a general setting so that details of a concrete data model would not obscure the picture. We now present such a minimalist model of incompleteness, following [51, 52].

5.1 Representation systems

To talk about incomplete information, one needs three key notions: of *objects*, of *complete objects*, which form a subset of the set of objects, and of *semantics* of incompleteness, which associates with every object a set of complete objects represented by it.

To formalize this, we consider triples $\langle \mathcal{D}, \mathcal{C}, [\![\,]\!] \rangle$, where

- $\mathcal{D}$ is a set of database objects (e.g., relational databases over the same schema),
- $\mathcal{C} \subseteq \mathcal{D}$ is the set of complete objects (e.g., databases without nulls);
- $[\![\,]\!]$ is a function from $\mathcal{D}$ to subsets of $\mathcal{C}$; the set $[\![x]\!] \subseteq \mathcal{C}$ is the semantics of object x.

We have not imposed any conditions at all, but some of them are needed to make this definition reflect the reality of incomplete data models. For instance, we expect a complete object c in the semantics of an incomplete object x to have more information than x. To express this, we fulfill our promise and bring the third line of work on incompleteness – based on orderings – into the picture. We define the *information ordering* as

$$x \preceq y \;\Leftrightarrow\; [\![y]\!] \subseteq [\![x]\!].$$

The intuition is that the more objects an incomplete object can potentially denote, the less information it contains (in the extreme case, if we have no information at all, every object is a possibility). We then impose two conditions on triples $\langle \mathcal{D}, \mathcal{C}, [\![\,]\!] \rangle$:

1. a complete object c denotes at least itself: $c \in [\![c]\!]$;
2. a complete object c is more informative than any incomplete object x it may represent: if $c \in [\![x]\!]$, then $x \preceq c$.

These conditions hold for $[\![\,]\!]_{\text{OWA}}$, $[\![\,]\!]_{\text{CWA}}$, and many other semantics of incompleteness.

We want to incorporate all approaches to incompleteness even at this general level, so we now bring in logical formulae. Let us assume that we have a set $\mathbb{F}$ of formulae and the satisfaction relation $\models$ between objects and formulae: $x \models \varphi$ means that φ is true in x. We write $\mathsf{Th}(x)$ for the *theory* of x:

$$\mathsf{Th}(x) \;=\; \{\varphi \mid x \models \varphi\}$$

is the set of formulae true in x. We write $\mathsf{Mod}(\varphi)$ for *models* of φ:

$$\mathsf{Mod}(\varphi) \;=\; \{x \mid x \models \varphi\}$$

is the set of all objects satisfying φ. These are extended to sets in the usual way:

$$\mathsf{Th}(X) = \bigcap_{x \in X} \mathsf{Th}(x) \quad \text{and} \quad \mathsf{Mod}(\Phi) = \bigcap_{\varphi \in \Phi} \mathsf{Mod}(\varphi).$$

Also, as before, we write $\mathsf{Mod}_{\mathsf{C}}(\varphi)$ for $\mathsf{Mod}(\varphi) \cap \mathcal{C}$.

Now we turn tuples $\langle \mathcal{D}, \mathcal{C}, [\![\,]\!], \mathbb{F} \rangle$ into *representation systems* that let us talk at once about objects, their semantics, logical representation, and information orderings. For that, we add the following conditions.

For formulae. Logical formulae must have enough power to define the semantics, as in (5), and must respect the informativeness of the objects.

Formally, for each object x there must be a formula δ_x so that $\mathsf{Mod}_{\mathsf{C}}(\delta_x) = [\![x]\!]$. Furthermore, $x \preceq y$ and $x \models \varphi$ imply $y \models \varphi$ for every formula φ. We also require that formulae be closed under conjunction.

For objects. Sets of objects cannot be too thin: thinking of relational databases, nulls should be replaceable by sufficiently many constants. That is, there must be sufficiently many valuations v of nulls so that $v(D) \in [\![D]\!]$; this is definitely true in standard semantics of incompleteness.

To state what 'sufficiently many' means, note that for every finite set $C \subset \mathsf{Const}$, we have an equivalence relation $\approx_C$ between databases: $D \approx_C D'$ says that there is an isomorphism f between D and D' that preserves constants in C (technically, both f and f^{-1} are the identity on C). Then, for every D, there is a valuation v so that $v(D) \approx_C D$. Indeed, we can just replace nulls with distinct constants outside of the finite set C. Thus, we have infinitely many equivalence relations $\approx_C$ such that for every database D, and every such relation, there is $D' \in [\![D]\!]$ so that $D' \approx_C D$.

Equivalence relations $\approx_C$ satisfy some basic properties. For instance, a formula that only mentions constants in C cannot distinguish two equivalent databases with respect to $\approx_C$. Also, if $D \approx_{C \cup C'} D$, then $D \approx_C D'$ and $D \approx_{C'} D'$.

These conditions can easily be formalized in our basic model. We assume that there is a family $\text{Iso} = \{\approx_j\}_{j \in J}$ of equivalence relations on $\mathcal{D}$ so that:

- The set $\{c \in [\![x]\!] \mid x \approx_j c\}$ is nonempty for each $x \in \mathcal{D}$ and $j \in J$;
- for all $j, j' \in J$, there is $k \in J$ so that $x \approx_k y$ implies both $x \approx_j y$ and $x \approx_{j'} y$; and
- for each formula $\varphi \in \mathbb{F}$, there must be $j \in J$ so that $x \approx_j y$ implies $x \models \varphi \Leftrightarrow y \models \varphi$.

Of course these three conditions hold for the family $\{\approx_C \mid C \text{ is a finite subset of } \mathsf{Const}\}$. From now on, we assume all the above conditions. We then call:

- $\mathbb{D} = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!], \text{Iso} \rangle$ a *domain*, and
- $\mathbb{RS} = \langle \mathbb{D}, \mathbb{F} \rangle$ a *representation system.*

We now give examples of those and show how they help us define the notion of certainty.

5.2 OWA and CWA representation systems

We now provide examples of representation systems corresponding to relational OWA and CWA semantics. Let $\mathcal{D}(\sigma)$ and $\mathcal{C}(\sigma)$ be the sets of all relational databases, and of all complete databases (not having nulls) of schema σ. The domains will be of the form $\mathbb{D}_*(\sigma) = \langle \mathcal{D}(\sigma), \mathcal{C}(\sigma), [\![\,]\!]_*, \text{Iso} \rangle$, where $*$ is OWA or CWA. The relations in Iso are, as seen earlier, of the form $\approx_C$ when C ranges over finite subsets of Const.

Under OWA, the set of formulae $\mathbb{F}$ can be taken to be UCQ, unions of conjunctive queries. Thus, $\mathbb{RS}_{\text{OWA}}(\sigma) = \langle \mathbb{D}_{\text{OWA}}(\sigma), \mathsf{UCQ} \rangle$ is a representation system under OWA. The formula δ_D is simply $\exists \bar{x}\ \text{PosDiag}(D)$, where $\text{PosDiag}(D)$, the positive diagram of D, is the conjunction of all atoms in D, where each null $\bot_i$ is associated with a variable x_i. For instance, if D contains a relation $R = \{(1,2), (2,\bot_1), (\bot_1,\bot_2)\}$, then $\text{PosDiag}(D) = R(1,2) \wedge R(2,x_1) \wedge R(x_1,x_2)$.

Under CWA, we used different features in the formula describing $[\![R]\!]_{\text{CWA}}$ in Section 4. Such formulae use universal quantification and implication, although in a limited way: the antecedent in implication was a relational atom. Such a class of formulae was already studied a long time ago [23]. Recall that *positive* FO formulae are those that do not use negation: they are formed from atomic formulae using $\wedge, \vee, \exists$, and $\forall$. We now extend this class to *positive formulae with universal guards*, denoted by $\mathsf{Pos}^{\forall\mathsf{G}}$. Such formulae are closed under $\wedge, \vee, \exists, \forall$ and the following rule: if $\varphi(\bar{x}, \bar{y})$ is a $\mathsf{Pos}^{\forall\mathsf{G}}$ formula in which all variables in $\bar{x}$ are distinct, and R is a relation symbol of the arity $|\bar{x}|$, then $\forall \bar{x}\ (R(\bar{x}) \rightarrow \varphi(\bar{x}, \bar{y}))$ is a $\mathsf{Pos}^{\forall\mathsf{G}}$ formula. In Section 6 we also describe this fragment in terms of relational algebra operators.

Then the CWA representation system is defined as $\mathbb{RS}_{\text{CWA}}(\sigma) = \langle \mathbb{D}_{\text{CWA}}(\sigma), \mathsf{Pos}^{\forall\mathsf{G}} \rangle$. For each D with $\mathsf{Null}(D) = \{\bot_1, \ldots, \bot_n\}$, the formula δ_D is

$$\exists x_1, \ldots, x_n \Big(\text{PosDiag}(D) \wedge \bigwedge_{R \in \sigma} \forall \bar{y}\ \big(R(\bar{y}) \rightarrow \bigvee_{\bar{t} \in R^D} \bar{y} = \bar{t}\big)\Big),$$

where the length of $\bar{y}$ and $\bar{t}$ is the arity of R, and $\bar{y} = \bar{t}$ means $\bigwedge_{i \leq \text{arity}(R)} (y_i = t_i)$.

We can also describe orderings $\preceq_{\text{OWA}}$ and $\preceq_{\text{CWA}}$ corresponding to $[\![\,]\!]_{\text{OWA}}$ and $[\![\,]\!]_{\text{CWA}}$. Recall that a *homomorphism* $h : D \rightarrow D'$, where D and D' are two databases of the same schema, is a mapping h from $\text{adom}(D)$ to $\text{adom}(D')$ so that $h(a) = a$ whenever $a \in \mathsf{Const}$, and for each tuple $\bar{t}$ in relation R of D, the tuple $h(\bar{t})$ is in the relation R of D', cf. [2, 7]. That is, h replaces nulls with either other nulls or constants, and leaves constants intact. A homomorphism is called *strong onto* if every tuple in D' is the image of a tuple in D, i.e., if $D' = h(D)$. Then [32, 51]:

- $D \preceq_{\text{OWA}} D' \quad \Leftrightarrow \quad \exists$ homomorphism $h : D \rightarrow D'$;
- $D \preceq_{\text{CWA}} D' \quad \Leftrightarrow \quad \exists$ strong onto homomorphism $h : D \rightarrow D'$.

The OWA and CWA semantics are not the only possible ones of course. For instance, one can use a weaker version of CWA, in which tuples can be added, as long as they do not add new elements to the active domain [59]. Then a representation system for this semantics will use the class of positive FO formulae, and the ordering is given by the existence of onto homomorphisms, which map $\text{adom}(D)$ onto $\text{adom}(D')$ [32, 52].

We can also connect representation systems and orderings. It can be shown that $\mathsf{Mod}(\delta_x) = \uparrow x = \{y \mid x \preceq y\}$, i.e., the set of all models of δ_x is the set of more informative objects.

5.3 Certainty in representation systems

Recall that to define certain answers to queries, we had to determine certain information contained in the set $Q([\![D]\!])$. Thus, the central problem for us is to understand how to define certainty contained in a set $X \subseteq \mathcal{D}$ of objects. With the dual view of objects as elements of an ordered set and as formulae, we have two approaches to defining certainty: as *knowledge* about the collection X, and as an *object* representing what is known about it. In general, the former is more flexible: we have already seen this in the example of conditional tables, which are just encodings of formulae. Trying to represent certainty as another object of the same kind can tie our hands too much, although in many important cases it can be done.

Certain information represented as knowledge The first attempt to describe with certainty information contained in a set X of objects is to find a formula φ so that $\mathsf{Mod}(\varphi) = X$. This is the approach of strong representation systems which look for an object A so that $[\![A]\!] = Q([\![D]\!])$; indeed, by the duality between formulae and objects, this is the same as requiring $\mathsf{Mod}_{\mathsf{C}}(\delta_A) = Q([\![D]\!])$. The problem is that not all sets X are of the form $\mathsf{Mod}(\varphi)$, for formula coming from logics of interest to us (of course we could use a highly expressive formalism but such a formalism would hardly be useful).

So following the approach of weak representation systems, we go for the next best thing, and replace equality by an equivalence relation. But equivalence between what? We now appeal to the duality again, and view X as a *theory*, i.e., $\mathsf{Th}(X)$, which says what we know about X with certainty in a given logical language. Indeed, $\mathsf{Th}(X)$ contains formulae φ which are true in *all* objects of X.

We now must find a formula φ representing this certain knowledge $\mathsf{Th}(X)$. Since two sets of formulae are equivalent if they have the same models, we need a formula φ such that $\mathsf{Mod}(\varphi) = \mathsf{Mod}(\mathsf{Th}(X))$. This is our certain knowledge of X, denoted by $\mathsf{certain}_{\mathcal{K}}X$. To summarize,

$$\mathsf{Mod}(\mathsf{certain}_{\mathcal{K}}X) \;=\; \mathsf{Mod}(\mathsf{Th}(X)). \qquad (6)$$

It is easy to show that if $\mathsf{Mod}(\varphi) = X$, or if $\mathsf{Mod}_{\mathsf{C}}(\varphi) = X$, then $\varphi = \mathsf{certain}_{\mathcal{K}}X$. Thus, (6) is a relaxation of the very strong notion of strong representation systems.

Certain information represented as object We appeal to the ordering-based approach to incompleteness. To represent what we know about X with certainty by an object y, this object must be less informative than any object $x \in X$ (as it reflects knowledge contained in all other objects in X as well). If we have two such objects y and y', and $y' \preceq y$, then of course we prefer y as it is giving us more information.

Thus, the object that we seek must be less informative than all objects in X, and at the same time the most informative among such objects. This is precisely the *greatest lower bound* of X, with respect to $\preceq$ (or $\bigwedge X$, using the standard order-theoretic notation). We denote it by $\mathsf{certain}_{\mathcal{O}}X$. To summarize,

$$\mathsf{certain}_{\mathcal{O}}X \;=\; \bigwedge X. \qquad (7)$$

A few remarks are in order. Neither $\mathsf{certain}_{\mathcal{K}}X$ nor $\mathsf{certain}_{\mathcal{O}}X$ need exist in general. When they exist, they may not be unique, but they are equivalent. That is, we may have different formulae φ and ψ satisfying (6) but they are equivalent: $\mathsf{Mod}(\varphi) = \mathsf{Mod}(\psi)$. Likewise, the greatest lower bound is not unique, but for every two objects x, x' satisfying the condition of being $\bigwedge X$ we have $x \preceq x'$ and $x' \preceq x$, which means $[\![x]\!] = [\![x']\!]$, i.e., x and x' are equivalent. The idea of using formula/object duality to define certain answers first appeared in [28], albeit in a very limited context, when $\mathbb{F} = \mathcal{D}$ and $x \models y$ was a shorthand for $y \preceq x$. The definitions we are using here, as well as the results below, are from [52].

These notions of certainty have some of the expected properties. For instance, the certain knowledge about $[\![x]\!]$ is δ_x, and its object representation is x itself: $\mathsf{certain}_{\mathcal{K}}[\![x]\!] = \delta_x$ and $\mathsf{certain}_{\mathcal{O}}[\![x]\!] = x$. In particular, $\mathsf{certain}_{\mathcal{O}}[\![x]\!] \models \mathsf{certain}_{\mathcal{K}}[\![x]\!]$, although in general $\mathsf{certain}_{\mathcal{O}}X \models \mathsf{certain}_{\mathcal{K}}X$ need not hold. Also the theory of all objects represented by x is the same as the theory of x, i.e., $\mathsf{Th}([\![x]\!]) = \mathsf{Th}(x)$.

Moreover, $\mathsf{certain}_{\mathcal{K}}X$ can be viewed as a greatest lower bound in a well-known ordering on formulae: implication $\psi \vdash \varphi$, which holds if every model of ψ is a model of φ. Thus, for a set of formulae Φ, we can look at its greatest lower bound in this preorder, denoted by $\bigwedge \Phi$. This is the most specific formula ψ that implies every $\varphi \in \Phi$ (i.e., every other formula that implies Φ must imply ψ as well). Note that since $\vdash$ is a preorder, technically $\bigwedge \Phi$ is a set of formulae, all of which, however, are equivalent.

Now the following is an alternative description of certain knowledge:

$$\mathsf{certain}_{\mathcal{K}}X \;=\; \bigwedge \mathsf{Th}(X). \qquad (8)$$

With this understanding of how to extract certain information, we are now going to apply it to sets $Q([\![D]\!])$, to see how certain answers must be defined.

6. MAKING CERTAIN ANSWERS EASY

We now want to use concepts from Section 5 to define certain answers to queries and to see when they can be computed efficiently, essentially using the existing technology. But first let us revisit the standard intersection-based notion of certain answers (1) to see what may go wrong if we use it. Take a very simple example: we have a database containing relation $R = \{(1,2),(2,\bot)\}$, and a query Q that just returns R. Following (1), $\mathsf{certain}(Q,R) = \{(1,2)\}$ under both OWA and CWA. This is, however, problematic for a number of reasons. First of all, such an answer misses information that there is a tuple whose first component is 2. Even more importantly, using intersection blindly for defining certain answers leads to counterintuitive results. Appealing to orderings describing the degree of incompleteness, we would expect $\mathsf{certain}(Q,R)$ not to exceed the level of informativeness of $Q(R')$ for each $R' \in [\![R]\!]$, as it presents information *common* to all such $Q(R')$. Under OWA, this is easily true, as $\{(1,2)\} \preceq_{\text{OWA}} R'$ for each $R' \in [\![R]\!]_{\text{OWA}}$. However, under CWA, exactly the opposite is true: $\{(1,2)\} \not\preceq_{\text{CWA}} R'$ for each $R' \in [\![R]\!]_{\text{CWA}}$. So in what sense $\{(1,2)\}$ is a certain answer under CWA is quite mysterious.

What causes this problem is the fact that intersection, in general, does *not* correspond to the ordering and the semantics of query answers. It may do so, but only under very limited conditions [52]. And if we want to define certainty as the greatest lower bound for the right ordering, as in (7) and (8), we need to understand what the orderings are. Note that we are defining orderings on query answers, so we are back to one of our basic questions: what is the semantics of query answers?

To answer this, we use a very basic principle:

> *if we know more about the input of a query,*
> *then we should know more about its output.*

However simple and natural this principle is, it is ignored by most approaches to incompleteness, starting with the definitions of strong and weak representation systems that somehow assume the same semantics for inputs and outputs.

6.1 Queries and naïve evaluation

To state results in a way independent of a particular data model, we view queries as mappings $Q : \mathbb{D} \to \mathbb{D}'$ between two domains $\mathbb{D} = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!], \text{Iso}\rangle$ and $\mathbb{D}' = \langle \mathcal{D}', \mathcal{C}', [\![\,]\!]', \text{Iso}'\rangle$. The basic principle stated above, that more informative inputs produce more informative outputs, is then the *monotonicity* of queries: if $x \preceq y$ then $Q(x) \preceq' Q(y)$, where $\preceq'$ is the ordering associated with the semantics $[\![\,]\!]'$. In particular, using blindly the same semantics for both databases and query results (as is often actually done) does not necessarily make sense.

Certain answer to Q on an object x represents certain information in $Q([\![x]\!])$. We have shown how to define it as knowledge, and as object, and the former requires a representation system $\mathbb{RS} = \langle \mathbb{D}', \mathbb{F}\rangle$ on query answers. In its presence, we can define certain answers as

$$\begin{array}{lrcl} \text{objects:} & \mathsf{certain}_{\mathcal{O}}(Q,x) & = & \mathsf{certain}_{\mathcal{O}} Q([\![x]\!]), \\ \text{knowledge:} & \mathsf{certain}_{\mathcal{K}}(Q,x) & = & \mathsf{certain}_{\mathcal{K}} Q([\![x]\!]). \end{array}$$

Then the following holds [52]: if Q is both monotone and generic, then

$$\mathsf{certain}_{\mathcal{O}}(Q,x) = Q(x) \qquad (9)$$

$$\mathsf{certain}_{\mathcal{K}}(Q,x) = \delta_{Q(x)} \qquad (10)$$

That is, naïve evaluation works: simply applying Q, without doing anything else, is what we need. By *genericity* we mean the standard notion of independence of query results under permutation: in the above setting, it is stated as follows: for every j, there is k so that $x \approx_k y$ implies $Q(x) \approx'_j Q(y)$. Relational queries under the standard interpretation of $\approx$ satisfy it.

This is great news: we can get properly defined certain answers without, seemingly, doing anything special. But a question remains: why do we need representation systems and (10) when we already have (9) at the level of objects? The answer is: because in the absence of a representation system, (9) need not be true; in fact, one first needs to establish (10) and then derive (9) as a corollary, as shown in [52].

Thus, if we want to rely on existing query evaluation techniques to produce correct answers in the presence of incompleteness, we need two things:

1. A proper semantics for query answers that ensures that more informative inputs produce more informative outputs; and
2. a representation system for that semantics.

We now show how to achieve these for relational databases under OWA and CWA.

6.2 Naïve evaluation under OWA and CWA

We now look at what naïve evaluation gives us when, as had been done before, we use the same semantics for query inputs and query answers; the semantics will be $[\![\,]\!]_{\text{OWA}}$ and $[\![\,]\!]_{\text{CWA}}$. Relational queries expressed in FO, or relational algebra, are guaranteed to be generic. Thus, we need to understand when they are monotone.

For now, look at a Boolean query Q, and the description of orderings $\preceq_{\text{OWA}}$ and $\preceq_{\text{CWA}}$. Then monotonicity simply means that if $D \models Q$ and $h : D \to D'$ is a homomorphism, for OWA (or strong onto homomorphism, for CWA), then $D' \models Q$. In other words, Q is *preserved under homomorphisms* (or strong onto homomorphisms).

Homomorphism preservation is a well known concept in logic; in particular, homomorphism preservation theorems give syntactic descriptions of classes of formulae satisfying these semantic conditions. Most of them are proved for infinite structures, but some work in the finite case too [20]. For instance, an FO sentence is preserved under homomorphisms iff it is equivalent to a union of conjunctive queries [63]. Preservation results for onto homomorphisms, however, only work in the infinite case, and are known to fail in the finite [6, 65]. Nonetheless, one can take advantage of sufficient conditions for preservation: for instance, it is known that the class of $\mathsf{Pos}^{\forall \mathsf{G}}$ formulae is preserved under strong onto homomorphisms [32].

We can now show when naïve evaluation works under OWA and CWA, if we use the same semantics for databases and query answers. Let Q be a query that is defined on databases of schema σ and produces databases of schema σ'. We say that $*$*-naïve evaluation works for* Q, where $*$ is OWA or CWA, if $\mathsf{certain}_{\mathcal{O}}(Q,D) =$

$Q(D)$, where we view Q as a mapping from $\mathbb{D}_*(\sigma)$ to $\mathbb{D}_*(\sigma')$ (remember that we need a semantics of answers specified in order to define the lower bound which gives us the notion of $\mathsf{certain}_{\mathcal{O}}$).

Then, combining (9) with the preservation result of [63] and the fact that UCQs form a representation system under OWA, we obtain:

- OWA-naïve evaluation works for UCQs, i.e., for positive relational algebra queries.

By combining (9) with the preservation result for $\mathsf{Pos}^{\forall\mathsf{G}}$ [32] and the fact that $\mathsf{Pos}^{\forall\mathsf{G}}$ forms a representation system under CWA, we obtain

- CWA-naïve evaluation works for $\mathsf{Pos}^{\forall\mathsf{G}}$ queries.

Can we get a better intuition of the power of $\mathsf{Pos}^{\forall\mathsf{G}}$ queries? Clearly they add something to the positive relational algebra, and clearly it cannot be the difference operator. Recall that many real-life queries with *for-all* conditions can be written using the *division* operator of relational algebra. If we have a relation R with attributes $A_1, \ldots, A_m, B_1, \ldots, B_k$ and a relation S with attributes $B_1, \ldots, B_k$, then $R \div S$ contains tuples of A-attributes of R that appear in R in every possible combination with a tuple from S, i.e., $R \div S = \{\bar{t} \in \pi_{\bar{A}}(R) \mid \forall \bar{s} \in S : (\bar{t}, \bar{s}) \in R\}$. Division is a derived operation of relational algebra, it can be expressed with $\sigma, \pi, \times, \cup, -$.

The easy way to think of the class $\mathsf{Pos}^{\forall\mathsf{G}}$ is that it adds to the positive relational algebra the operation of diving by a base relation (i.e., S in $R \div S$ must be a relation in the database). In fact, the class is slightly more expressive, and defined as follows.

Let Δ be the query returning $\{(a, a) \mid a \in \text{adom}(D)\}$; it is easily definable in positive relational algebra. Let $\mathsf{RA}(\Delta, \pi, \times, \cup)$ be the class of relational algebra queries obtained from base relations and Δ by closing them under π, $\times$, and $\cup$. Now we define RA_{CWA} as follows:

- Each relation name is an RA_{CWA} query;
- RA_{CWA} is closed under $\sigma, \pi, \times$, and $\cup$ (i.e., all operations except difference);
- if Q is an RA_{CWA} query, and Q' is an $\mathsf{RA}(\Delta, \pi, \times, \cup)$ query, then $Q \div Q'$ is in RA_{CWA}.

One can then show the following.

- $\mathsf{RA}_{\text{CWA}} = \mathsf{Pos}^{\forall\mathsf{G}}$, and consequently:
- CWA-naïve evaluation works for RA_{CWA}.

Thus, we have two large classes of queries for which simply evaluating queries using existing technology does produce correct answers.

7. WHERE DO WE GO FROM HERE?

Summary

We started with a rather bleak assessment of the practical use of nulls: *"If you have any nulls in your database, you're getting wrong answers to some of your queries. What's more, you have no way of knowing, in general, just which queries you're getting wrong answers to; all results become suspect. You can never trust the answers you get from a database with nulls"* [24]. We then analyzed the state of theoretical research on incompleteness in databases, and concluded that it was in disarray as well.

But by answering some of the questions posed at the end of Section 3, we can alleviate some of the fears expressed in [24]. Not *all* results become suspect. You can *sometimes* trust the result you get from a database with nulls. One precondition for it is to have naïve, or marked nulls, but this, as mentioned already, is doable, and implemented in existing DBMS [38]. Then one can fully trust answers to positive relational algebra queries, even extended with a rather liberal use of the division operator under the closed-world semantics. In general, if the semantics of the answers is right, one *can* trust the answers provided by the standard query evaluation.

As for the key questions about the state of the theory of incompleteness, we provided a few answers. We argued that the semantics of query answers need not be the same as the semantics of input databases, and that it should be chosen in a way that more informative inputs provide more informative outputs. Intersection is not the only way to define certain answers – in fact sometimes it is a plain wrong way to define certain answers. We should be open to viewing query answers as both objects and a representation of knowledge about all possible answers. And we should be free to go back and forth between several paradigms for dealing with incompleteness: the standard object-based view, the logical-theory view, and the ordering view.

Future directions

More expressive queries We have seen how to handle positive relational algebra queries and their extension with the division operator. Can we push this further? Of course yes: (9) and (10) tell us that there is no limit as long as the semantics is chosen correctly. So the next obvious step is to analyze semantic requirements for different types of queries, and see when they can be used together with the standard query evaluation techniques.

Evaluation techniques Naïve evaluation simply applies existing query evaluation as is, and it produces the right answers under the right semantics. But we are assuming we actually know how to apply a given query to a database with nulls, and this need not always be the case. There are a variety of possible evaluation techniques that need to be investigated. Returning to our example from the introduction, it is quite bad that the query says no payments are missing, but at least we are not chasing good guys – there are no false positives. Can this always be guaranteed? Sound evaluation has been addressed before [61], but we do not fully understand efficient query evaluation techniques with nulls and their interaction with the right notions of certainty. The logical approach to incomplete databases also fits in well with the three-value semantics of SQL: theories representing databases need not be complete and may lead to unknown answers [42, 60]. One can also consider applying existing reasoning procedures with unknown outcomes (e.g., [48]) to databases with nulls.

Handling constraints This subject has been addressed, in particular for functional dependencies, with several different attempts to define when an incomplete database satisfies a constraint, see, e.g., [12, 46]. Some of these results were also applied to database design, both relational and beyond [8, 47]. However, unlike in the case of querying, little attention has been paid to the role of the semantics. But constraints are queries, after all, and we should be able to apply techniques developed here to their study. Most constraints, however, involve universal quantification, and thus special care needs to be taken in evaluating them.

Beyond relations: XML and graphs We have looked only at relational databases but of course there are other models of data. Already a long time ago, analogs of the basic concepts from [40] were worked out for nested relations [45]. More recently, there was some activity in studying incompleteness in XML [4, 13, 27, 28]. One of the key differences is that not only data but also some structural features can be missing, although it was shown that structural incompleteness leads to intractability very quickly. Most of these papers used XML-to-relations queries to define certain answers by means of intersection; an exception is [28] in which a rudimentary version of (6) was given. To extend our techniques to XML, we would need to find classes of queries satisfying homomorphism preservation conditions, which is not a trivial task at all for tree-like structures [11] and it becomes even harder with data present. A few initial results in this direction were very recently reported in [31]. And in the case of graph databases we know even less; see [14, 56] for attempts at defining incompleteness over graph data and RDF.

Applications Some of the most important applications of incomplete data occur in tasks such as data integration, data exchange, and consistent query answering: in fact, in all three of them, the standard semantics of query answering is based on certain answers. The need to apply techniques from incomplete databases in these areas is well recognized, see, e.g., [1, 35, 41] for data integration and [10, 36, 50] for data exchange. However, in all the cases, the standard intersection-based definition is used, and with few exceptions, OWA is the dominating semantics. In fact quite often naïve evaluation is used for query answering in cases where it is known not to work (as explained in [7, 50]). It thus seems to be a natural next step to apply the notions of certainty we presented here in these applications. This may well involve rethinking several of the semantic assumptions made in the past.

Acknowledgments I am very grateful to Marcelo Arenas, Pablo Barceló, Diego Figueira, Amélie Gheerbrant, Filip Murlak, Juan Reutter, and Cristina Sirangelo for reading earlier drafts and providing many helpful comments. Thanks also to Serge Abiteboul, Claire David, Giuseppe de Giacomo, Maurizio Lenzerini, Wim Martens, Antonella Poggi, Lucian Popa, and Domagoj Vrgoč for comments, discussions, and questions reflected in this paper. This work is partly supported by EPSRC grant J015377.

8. REFERENCES

[1] S. Abiteboul, O. Duschka. Complexity of answering queries using materialized views. In *PODS'98*, pages 254–263.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[3] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78(1):158–187, 1991.

[4] S. Abiteboul, L. Segoufin, and V. Vianu. Representing and querying XML with incomplete information. *ACM TODS*, 31(1):208–254, 2006.

[5] S. Abiteboul et al. *Web Data Management.* Cambridge University Press, 2011.

[6] M. Ajtai and Y. Gurevich. Monotone versus positive. *J. ACM*, 34(4):1004–1015, 1987.

[7] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Foundations of Data Exchange.* Cambridge University Press, 2014.

[8] M. Arenas and L. Libkin. A normal form for XML documents. *ACM TODS*, 29(1):195–232, 2004.

[9] M. Arenas and L. Libkin. XML data exchange: Consistency and query answering. *J. ACM*, 55(2), 2008.

[10] M. Arenas, J. Pérez, and J. L. Reutter. Data exchange beyond complete data. *J. ACM*, 60(4), 2013.

[11] A. Atserias, A. Dawar, and P. Kolaitis. On preservation under homomorphisms and unions of conjunctive queries. *J. ACM*, 53(2):208–237, 2006.

[12] P. Atzeni and N. M. Morfuni. Functional dependencies in relations with null values. *IPL*, 18(4):233–238, 1984.

[13] P. Barceló, L. Libkin, A. Poggi, and C. Sirangelo. XML with incomplete information. *J. ACM*, 58(1), 2010.

[14] P. Barceló, L. Libkin, and J. Reutter. Querying regular graph patterns. *J. ACM*, 61(1), 2014.

[15] L. Bertossi. *Database Repairing and Consistent Query Answering.* Morgan&Claypool Publishers, 2011.

[16] P. Buneman, A. Jung, A. Ohori. Using powerdomains to generalize relational databases. *TCS*, 91(1):23–55, 1991.

[17] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*, pages 260–271, 2003.

[18] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Semi-structured data with constraints and incomplete information. In *Description Logics*, 1998.

[19] A. K. Chandra, P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC'77*, pages 77–90.

[20] C. Chang, H. Keisler. *Model Theory.* North Holland, 1990.

[21] E. F. Codd. Understanding relations (installment #7). *FDT - Bulletin of ACM SIGMOD*, 7(3):23–28, 1975.

[22] E. F. Codd. Extending the database relational model to capture more meaning. *ACM TODS*, 4(4):397–434, 1979.

[23] K. Compton. Some useful preservation theorems. *Journal of Symbolic Logic*, 48(2):427–440, 1983.

[24] C. J. Date. *Database in Depth - Relational Theory for Practitioners.* O'Reilly, 2005.

[25] C. J. Date. A critique of Claude Rubinson's paper 'Nulls, three-valued logic, and ambiguity in SQL: critiquing Date's critique'. *SIGMOD Record*, 37(3):20–22, 2008.

[26] C. J. Date and H. Darwin. *A Guide to the SQL Standard.* Addison-Wesley, 1996.

[27] C. David, A. Gheerbrant, L. Libkin, and W. Martens. Containment of pattern-based queries over data trees. In *ICDT*, pages 201–212, 2013.

[28] C. David, L. Libkin, and F. Murlak. Certain answers for XML queries. In *PODS*, pages 191–202, 2010.

[29] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theoretical Computer Science*, 336:89–124, 2005.

[30] W. Fan and F. Geerts. *Foundations of Data Quality Management.* Morgan&Claypool Publishers, 2012.

[31] D. Figueira and L. Libkin. Pattern logics and auxiliary relations. In *LICS*, 2014.

[32] A. Gheerbrant, L. Libkin, and C. Sirangelo. When is naïve evaluation possible? In *PODS*, pages 75–86, 2013.

[33] A. Gheerbrant, L. Libkin, and T. Tan. On the complexity of query answering over incomplete XML documents. In *ICDT*, pages 169–181, 2012.

[34] G. Grahne. *The Problem of Incomplete Information in Relational Databases.* Springer, 1991.

[35] G. Grahne. Information integration and incomplete information. *IEEE Data Eng. Bull.*, 25(3):46–52, 2002.

[36] G. Grahne and A. Onet. Representation systems for data exchange. In *ICDT*, pages 208–221, 2012.

[37] J. Grant. Null values in a relational data base. *Inf. Process. Lett.*, 6(5):156–157, 1977.

[38] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. In *SIGMOD*, pages 805–810, 2005.

[39] A. Halevy. Theory of answering queries using views. *SIGMOD Record*, 29(1):40–47, 2000.

[40] T. Imielinski and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[41] D. Lembo, M. Lenzerini, and R. Rosati. Source inconsistency and incompleteness in data integration. In *Description Logics*, 2002.

[42] M. Lenzerini. Type data bases with incomplete information. *Inf. Sci.*, 53(1-2):61–87, 1991.

[43] M. Lenzerini. Data integration: a theoretical perspective. In *PODS*, pages 233–246, 2002.

[44] N. Lerat and W. L. Jr. Nonapplicable nulls. *Theor. Comput. Sci.*, 46(3):67–82, 1986.

[45] M. Levene and G. Loizou. Semantics for null extended nested relations. *ACM TODS*, 18(3):414–459, 1993.

[46] M. Levene and G. Loizou. Axiomatisation of functional dependencies in incomplete relations. *Theor. Comput. Sci.*, 206(1-2):283–300, 1998.

[47] M. Levene and G. Loizou. Database design for incomplete relations. *ACM TODS*, 24(1):80–125, 1999.

[48] H. J. Levesque. A completeness result for reasoning with incomplete first-order knowledge bases. In *KR*, pages 14–23, 1998.

[49] L. Libkin. A semantics-based approach to design of query languages for partial information. In *Semantics in Databases*, LNCS vol. 1358, pages 170–208. Springer, 1995.

[50] L. Libkin. Data exchange and incomplete information. In *PODS*, pages 60–69, 2006.

[51] L. Libkin. Incomplete information and certain answers in general data models. In *PODS*, pages 59–70, 2011.

[52] L. Libkin. Certain answers as objects and knowledge. In *Principles of Knowledge Representation and Reasoning (KR)*, 2014.

[53] L. Libkin and C. Sirangelo. Data exchange and schema mappings in open and closed worlds. *J. Comput. Syst. Sci.*, 77(3):542–571, 2011.

[54] W. Lipski. On semantic issues connected with incomplete information databases. *ACM TODS*, 4(3):262–296, 1979.

[55] B. Marnette, G. Mecca, P. Papotti, S. Raunich, and D. Santoro. ++Spicy: an opensource tool for second-generation schema mapping and data exchange. *PVLDB*, 4(12):1438–1441, 2011.

[56] C. Nikolaou and M. Koubarakis. Incomplete information in RDF. In *RR*, pages 138–152, 2013.

[57] A. Ohori. Semantics of types for database objects. *Theoretical Computer Science*, 76:53–91, 1990.

[58] R. Reiter. On closed world data bases. In *Logic and Data Bases*, pages 55–76, 1977.

[59] R. Reiter. Equality and domain closure in first-order databases. *J. ACM*, 27(2):235–249, 1980.

[60] R. Reiter. Towards a logical reconstruction of relational database theory. In *On Conceptual Modelling*, pages 191–233, 1982.

[61] R. Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *J. ACM*, 33(2):349–347, 1986.

[62] R. Rosati. On the decidability and finite controllability of query processing in databases with incomplete information. In *PODS*, pages 356–365, 2006.

[63] B. Rossman. Homomorphism preservation theorems. *J. ACM*, 55(3), 2008.

[64] B. Rounds. Situation-theoretic aspects of databases. In *Situation Theory and Applications*, volume 26 of *CSLI*, pages 229–256. 1991.

[65] A. Stolboushkin. Finitely monotone properties. In *LICS*, pages 324–330, 1995.

[66] R. van der Meyden. Logical approaches to incomplete information: A survey. In *Logics for Databases and Information Systems*, pages 307–356, 1998.

[67] C. Zaniolo. Database relations with null values. *J. Comput. Syst. Sci.*, 28(1):142–166, 1984.