How to Define Certain Answers*

Leonid Libkin
University of Edinburgh

Abstract

The standard way of answering queries over incomplete databases is to compute *certain answers*, defined as the intersection of query answers on all complete databases that the incomplete database represents. But is this universally accepted definition correct? We argue that this "one-size-fits-all" definition can often lead to counterintuitive or just plain wrong results, and propose an alternative framework for defining certain answers.

We combine three previously used approaches, based on the semantics and representation systems, on ordering incomplete databases in terms of their informativeness, and on viewing databases as knowledge expressed in a logical language, to come up with a well justified and principled notion of certain answers. Using it, we show that for queries satisfying some natural conditions (like not losing information if a more informative input is given), computing certain answers is surprisingly easy, and avoids the complexity issues that have been associated with the classical definition.

1 Introduction

Handling incomplete information is one of the oldest topics in database research. It has been tackled both from the database perspective, resulting in classical notions of the semantics and complexity of query evaluation [Abiteboul *et al.*, 1991; Imielinski and Lipski, 1984], and from the AI perspective, providing an alternative view of the problem, see, e.g., [Reiter, 1982; Lenzerini, 1991]. With applications increasingly focusing on large amounts of heterogeneous data, the problem of incomplete information is becoming much more pronounced. It appears in many important application areas such as data integration [Lenzerini, 2002], data exchange [Arenas *et al.*, 2014], inconsistent databases [Bertossi, 2011], probabilistic data [Suciu *et al.*, 2011], query answering using ontologies [Kontchakov *et al.*, 2011], etc.

To answer queries in the presence of incompleteness, one looks for *certain answers*: those that do not depend on the interpretation of unknown data. The concept was defined over 35 years ago [Grant, 1977; Lipski, 1979] as follows. Assume that the semantics $[\![D]\!]$ of an incomplete database D is given as the set of all complete databases (or possible worlds) D' which D can represent. If a query Q returns a set of objects (e.g., a set of tuples for relational databases), the certain answer to Q on D is defined as

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

This definition has been universally applied to all the semantics of incompleteness, and in all the application scenarios:

- in incomplete relational, or XML, or graph data, D is an incomplete database, and $[\![D]\!]$ contains all databases D obtained by interpreting the incomplete features of D, e.g., replacing nulls with constant values [Abiteboul *et al.*, 1995; Barceló *et al.*, 2010; Barceló *et al.*, 2014];
- in data integration or data exchange, D is a set of data sources available to us, and $[\![D]\!]$ contains all databases D' that, together with D, satisfy the conditions of a schema mapping that guides the integration/exchange process [Lenzerini, 2002; Arenas *et al.*, 2014];
- in consistent query answering, D is an inconsistent database, and $[\![D]\!]$ contains all the minimal repairs that restore the integrity of the database [Bertossi, 2011];
- in query answering over ontologies, D is a database, and $[\![D]\!]$ contains all D' that expand D and satisfy all ontology constraints [Kontchakov *et al.*, 2011].

In all the cases, one then uses (1) to define certain answers. The intuition is that this gives us the set of tuples independent of the interpretation of the missing information in D. Such a universal adoption of this basic definition has another consequence: certain answers themselves contain no missing information. In fact many algorithms for computing certain answers have, as the last step, elimination of any objects (say, rows in relational databases) with missing data.

The question that we address here is the following: *Is this standard "one-size-fits-all" definition really the right one to use for all the semantics, and all the applications?* The answer, as we shall argue, is negative: the standard intersection

*This paper was invited for submission to the Best Papers From Sister Conferences Track, based on a paper that appeared in KR 2014.

semantics, as well as the assumption that no missing information is present in the answers, leads to many problems, and crucially to producing meaningless query answers.

To argue that this is the case, and to explain some basic ideas behind the alternative approach we propose, note that in the database field, one tends to operate with *objects* (i.e., relations, XML documents, graph databases, etc.); in particular, queries take objects and return objects. Thus, the idea behind certain answers is to find an *object* A representing the *set of objects* $Q(\llbracket D \rrbracket) = \{Q(D') \mid D' \in \llbracket D \rrbracket\}$. Such an object must contain information common to all the objects in $Q(\llbracket D \rrbracket)$: that is, it must be no more informative than any of the objects in $Q(\llbracket D \rrbracket)$.

Now take a simple example: we have a relation $R = \{(1,2),(3,\bot)\}$ in a database, where $\bot$ represents a null, or a missing value. The query Q returns R itself. Then $\mathsf{cert}_\cap(Q,D) = \{(1,2)\}$ under every reasonable semantics of incompleteness. But is it less informative than all of $Q(D')$ for $D' \in \llbracket D \rrbracket$? Not necessarily. Consider the very common *closed-world* semantics of incompleteness. Then $(1,2)$ is not less informative than any of the answers $Q(D')$ for $D' \in \llbracket D \rrbracket$ which are of the form $\{(1,2),(3,n)\}$ for different values n. Indeed, under the closed world semantics, the answer $\{(1,2)\}$ contains *additional* information that no tuple except $(1,2)$ is present. Thus, returning just $(1,2)$ in this case makes no sense at all. In fact, deleting a tuple, as $\mathsf{cert}_\cap$ forces us to do, eliminates *data*, but not *information*, from certain answers.

The problem with (1) is even more visible under the approach, pioneered by [Reiter, 1982], that views databases as logical theories and query answering as logical implication. The fact $R(1,2)$ is certainly implied by the database. But it is *not* the only fact that is implied; we can also deduce $\exists x\ R(1,2) \wedge R(3,x)$ with certainty, as well as $\exists x \forall y\ (y = 1 \vee y = 2 \vee y = 3 \vee y = x)$ under the closed-world semantics, since we cannot expand the database.

There are a few lessons we learn from this simple example. First, certain answers can be presented as both objects and logical formulae. Second, they depend on both logical languages and semantics used, and third, taking intersection and removing missing values from the answers is not always the right way to compute them.

Our goal is to develop an alternative framework for handling certain answers to queries. For that, we combine the approaches to viewing databases as objects [Imielinski and Lipski, 1984] and as logical theories [Reiter, 1982], with the idea of ordering incomplete databases based on their informativeness [Buneman *et al.*, 1991]. Specifically, the key elements of our framework are as follows.

- Certain answers can be viewed as either objects or theories, depending on the semantics, and the logical formalism used. The former is in line with the standard database approach, while the latter defines *certain knowledge* about query answers over incomplete databases.
- Both ways are based on extracting certain information from a set of objects. Each way defines certainty as a greatest lower bound: either of a set of objects, or the *theory* of that set of objects, with the ordering meaning "being more informative".
- Proper query answering is based on the notion of *representation systems*: these are a natural relaxation of a rather restrictive concept of what database people call strong representation systems. Representation systems let one define important sets of objects by logical formulae, in the spirit of [Reiter, 1982].
- Under the choice of the right semantics for both query inputs and query answers, certain answers – as both objects and knowledge – can be found by straightforward database query evaluation. Thus, with the correct choice of semantics and representation system, we can use *existing* query evaluation techniques for obtaining correct answers in the presence of incomplete information.

Most of the results are shown in an abstract setting, for two reasons. First, it makes them applicable to other data models, beyond relational databases. Second, it helps us see the essential conditions that need to be imposed on queries and the semantics of incompleteness, without being too "clouded" by details of a particular data model. At the same time we use two common relational semantics – open and closed world – to translate general results into concrete examples.

The goal of this paper is to explain the main idea of the new approach to defining certain answers. It omits some of the technical development (in particular, proofs of results, and additional examples) that can be found in [Libkin, 2014a] and also in [Libkin, 2014b].

2 Incompleteness in relational databases

We recall some basic definitions and facts about incomplete information in relational databases. Such databases are populated by two types of values: *constants* (e.g., $1, 2, \ldots$) and *nulls*. We thus assume countably infinite sets of constants, denoted by Const, and of nulls, denoted by Null. Nulls themselves are denoted by $\bot$, sometimes with sub- or superscripts.

A relational *vocabulary* (often called *schema* in database literature) is a set of relation names with associated arities. An incomplete relational instance D assigns to each k-ary relation symbol R from the vocabulary a k-ary relation R^D over $\mathsf{Const} \cup \mathsf{Null}$, i.e., a finite subset of $(\mathsf{Const} \cup \mathsf{Null})^k$. If the instance D is clear from the context, we may write R instead of R^D. Sets of constants and nulls that occur in D are denoted by $\mathsf{Const}(D)$ and $\mathsf{Null}(D)$. The *active domain* of D is $\mathsf{adom}(D) = \mathsf{Const}(D) \cup \mathsf{Null}(D)$. A *complete* database D has no nulls, i.e., $\mathsf{adom}(D) \subseteq \mathsf{Const}$.

A *valuation* of nulls on an incomplete database D is a map $v : \mathsf{Null}(D) \to \mathsf{Const}$ assigning a constant value to each null. It naturally extends to databases, so we can write $v(D)$ as well. The standard semantics of incompleteness in relational databases are defined in terms of valuations, see [Abiteboul *et al.*, 1995; Imielinski and Lipski, 1984]. These are the *closed world assumption*, or CWA semantics:

$$\llbracket D \rrbracket_{\text{CWA}} = \{v(D) \mid v \text{ is a valuation}\},$$

and the *open-world assumption*, or OWA semantics:

$$\llbracket D \rrbracket_{\text{OWA}} = \left\{ D' \;\middle|\; \begin{array}{l} D' \text{ is complete and} \\ v(D) \subseteq D' \text{ for some valuation } v \end{array} \right\}.$$

That is, under CWA, we simply instantiate nulls by constants; under OWA, we can also add arbitrary tuples.

Given an incomplete database D, a semantics of incompleteness $[\![\,]\!]$, and a query Q, the standard notion of certain answers under $[\![\,]\!]$ is defined by (1), i.e., $\mathsf{cert}_\cap(Q,D) = \bigcap\{Q(R) \mid R \in [\![D]\!]\}$.

Most of database query languages are based on *first-order predicate logic*, or FO, whose formulae are built from relational atoms $R(\bar{x})$, where R is a vocabulary symbol, equational atoms $x = y$, and are closed under Boolean connectives $\wedge, \vee, \neg$ and quantifiers $\exists$ and $\forall$.

The fragment that disallows $\neg$ and $\forall$ (i.e., has $\wedge, \vee, \exists$) is referred to as *existential positive formulae*, denoted by $\exists\mathsf{Pos}$. In terms of their expressiveness, they correspond precisely to *unions of conjunctive queries* (although $\exists\mathsf{Pos}$ formulae can be more compact).

The fragment without negation (i.e., the $\wedge, \vee, \exists, \forall$ fragment) is referred to as *positive formulae*, denoted by Pos.

Finding certain answers to FO queries ranges from coNP-complete for CWA to undecidable under OWA, see [Abiteboul *et al.*, 1991]. However, sometimes $\mathsf{cert}_\cap(Q,D)$ can be obtained by almost straightforward query evaluation, namely by evaluating $Q(D)$ and then throwing away the tuples with nulls. We shall denote this by $Q^{\mathsf{C}}(D)$. For instance, if the query Q just returns a relation R, and $R^D = \{(1,2),(1,\bot)\}$, then both $Q^{\mathsf{C}}(D)$ and $\mathsf{cert}_\cap(Q,D)$ are $\{(1,2)\}$. In fact, $\mathsf{cert}_\cap(Q,D)$ and $Q^{\mathsf{C}}(D)$ coincide for $\exists\mathsf{Pos}$ queries under both OWA and CWA [Imielinski and Lipski, 1984].

3 Objects and knowledge

The key idea is to decouple objects and knowledge about them, expressed as their description in terms of some logical formalism. We want to do this at the highest level of abstraction, so that the framework would not be limited to just relational databases, but instead would be applicable across multiple data models. For this, we use a very minimalist setting inspired by abstract model theory [Barwise and Feferman, 1985] or information systems [Gunter, 1992], with some specific features tailored to handle incompleteness, as also used in [Libkin, 2011; Gheerbrant *et al.*, 2014].

We have two basic entities: (database) *objects*, and *formulae* they satisfy. Objects themselves are either incomplete or complete, and each object has its semantics defined as the set of more informative complete objects.

To formalize this, we define a *database pre-domain* as a triple $\mathbb{D}^\circ = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!] \rangle$, where

- $\mathcal{D}$ is a set of database objects (e.g., relational databases over the same schema),
- $\mathcal{C}$ 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 an object x.

We next introduce *information ordering*:

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

The idea is simple: the more we know about an object, the fewer objects it can denote. For instance, if we know nothing about it, it can denote any object whatsoever; getting additional information reduces the set of possible worlds.

We assume that every pre-domain satisfies two conditions, which in fact are immediate in the standard semantics of incompleteness:

- a complete object denotes at least itself: if $c \in \mathcal{C}$, then $c \in [\![c]\!]$;
- if c is a possible world for x, then we know at least as much about c as we know about x: if $c \in [\![x]\!]$, then $x \preceq c$.

Next, we add *knowledge* about objects. A *pre-representation system* is a triple $\mathbb{RS}^\circ = \langle \mathbb{D}^\circ, \mathbb{F}, \models \rangle$, where

- $\mathbb{D}^\circ$ is a pre-domain;
- $\mathbb{F}$ is a set of *formulae*, and
- $\models$ is the satisfaction relation, i.e., a subset of $\mathcal{D} \times \mathbb{F}$ such that $x \preceq y$ and $x \models \varphi$ imply $y \models \varphi$.

The intuition is that formulae in $\mathbb{F}$ express knowledge we possess about objects in $\mathcal{D}$, and if we know something about an object, we also know it about a more informative object.

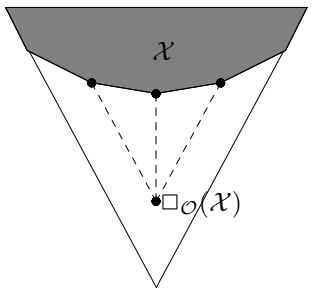
We shall write $\mathsf{Th}(x)$ for the *theory* of x, i.e., $\{\varphi \mid x \models \varphi\}$ and $\mathsf{Mod}(\varphi)$ for models of φ, i.e., $\{x \mid x \models \varphi\}$. These are extended to sets in the usual way: $\mathsf{Th}(X) = \bigcap_{x\in X} \mathsf{Th}(x)$ and $\mathsf{Mod}(\Phi) = \bigcap_{\varphi\in\Phi} \mathsf{Mod}(\varphi)$.

3.1 Certain information

Computing certain answers boils down to finding certain information contained in a set of objects; in the case of query answering, in $Q([\![D]\!]) = \{Q(D') \mid D' \in [\![D]\!]\}$. Thus, we need to know how to define certain information contained in a set of objects $X \subseteq \mathcal{D}$. The usual database approach is to represent this information as another object, but of course we argue that it can be viewed as both object and knowledge.

Certain information as object If we want 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 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). If it is exists, we denote it by $\Box_{\mathcal{O}} X$. This is illustrated by the picture below.



Certain information as knowledge We want to describe X by a single formula summarizing what we know about it with certainty. If $X = \mathsf{Mod}(\varphi)$, then φ is such a formula, but generally, X need not be of the form $\mathsf{Mod}(\varphi)$.

So we go for the next best thing: we want a formula that is equivalent to the *theory* of X. Indeed, $\mathsf{Th}(X)$ is what we know about X with certainty, and we want to capture this by a formula. Two sets of formulae are equivalent when they have the same models, so a formula equivalent to the theory of X is a formula φ such that $\mathsf{Mod}(\varphi) = \mathsf{Mod}(\mathsf{Th}(X))$. If it exists, we denote it by $\Box_{\mathcal{K}} X$.

Thus, certain information contained in X is described as:

- At the object level as $\Box_{\mathcal{O}} X = \bigwedge X$; and
- At the knowledge level as a formula $\Box_{\mathcal{K}} X$ so that $\mathsf{Mod}(\Box_{\mathcal{K}} X) = \mathsf{Mod}(\mathsf{Th}(X))$.

Note that neither $\Box_{\mathcal{O}} X$ nor $\Box_{\mathcal{K}} X$ need exist in general (in fact it is easy to come up with examples of preorders without greatest lower bounds). Even if they exist, they need not be unique. This is not an issue, however, as they are *equivalent*. Since $\preceq$ is a preorder, the greatest lower bound is, technically speaking, a set of objects, but every two such objects y, y' are equivalent: $y \preceq y'$ and $y' \preceq y$, and thus $[\![y]\!] = [\![y']\!]$. If we have multiple formulae φ for which $\mathsf{Mod}(\varphi) = \mathsf{Mod}(\mathsf{Th}(X))$, then every two such formulae φ, φ' are equivalent: $\mathsf{Mod}(\varphi) = \mathsf{Mod}(\varphi')$. So we shall write $y = \Box_{\mathcal{O}} X$ or $\varphi = \Box_{\mathcal{K}} X$, meaning y or φ is one of the equivalent objects or formulae.

Example Consider the example from the introduction, of a database D containing $(1, 2)$ and $(3, \bot)$ in a relation R. if $X = [\![D]\!]_{\text{OWA}}$, then $\Box_{\mathcal{O}} X$ is just D itself, as expected. If $\mathbb{F} = \exists\mathsf{Pos}$, then $\Box_{\mathcal{K}} X = \exists z\, R(1, 2) \wedge R(3, z)$. If $\mathbb{F}$ is the set of ground facts and their conjunctions, then $\Box_{\mathcal{K}} X = R(1, 2)$.

4 Representation systems

To make pre-representation systems capture realistic scenarios of dealing with incomplete information, we must impose conditions saying, essentially, that the sets of objects and formulae are not too "thin": there are enough complete objects, and there are formulae defining some basic sets of objects.

There are enough objects To motivate this condition, consider a database D with a single tuple $(\bot, \bot)$. In its semantics, we are allowed to replace this tuple by an arbitrary tuple (c, c) with $c \in \mathsf{Const}$. Moreover, the resulting database is isomorphic to the original one: for instance, they agree on logical formulae not mentioning constants. Even if we have a logical formula mentioning constants from a finite set C, we can replace $\bot$ by constants outside this set C: this is what we mean by the existence of enough complete objects. Then D and the result of the replacement will still agree on formulae that only refer to constants in C.

To formalize this, define a *database domain* $\mathbb{D}$ as a tuple $\langle \mathcal{D}, \mathcal{C}, [\![\,]\!], \text{Iso}\rangle$ where $\langle \mathcal{D}, \mathcal{C}, [\![\,]\!]\rangle$ is a pre-domain, and Iso is a family $\{\approx_j\}_{j \in J}$ of equivalence relations on $\mathcal{D}$ so that:

- The set $[\![x]\!]_{\approx_j} = \{c \in [\![x]\!] \mid x \approx_j c\}$ is nonempty for each $x \in \mathcal{D}$ and $j \in J$;
- for every $j, j' \in J$, there is $k \in J$ so that $x \approx_k y$ implies $x \approx_j y$ and $x \approx_{j'} y$.

For relational databases, $j \in J$ enumerate finite sets of constants C_j, and $D \approx_j D'$ means that there is an isomorphism between D and D' preserving constants in C_j. The first condition says that we can replace nulls by constants outside C_j (since $\mathsf{Const} - C_j$ is infinite), and the second one says that $C_k = C_j \cup C_{j'}$ preserves constants in both C_j and $C_{j'}$.

There are enough formulae We assume that formulae are closed under conjunction. We also assume that $[\![x]\!]$ can be described by a formula. That is, for each object x, there is a formula δ_x in $\mathbb{F}$, such that $\mathsf{Mod}(\delta_x) = {\uparrow}x$ (this is equivalent to $\mathsf{Mod}(\delta_x) \cap \mathcal{C} = [\![x]\!]$).

When all these conditions are satisfied, we say that the triple $\mathbb{RS} = \langle \mathbb{D}, \mathbb{F}, \models\rangle$ is a *representation system* if for each $\varphi \in \mathbb{F}$, there is $j \in J$ so that $x \models \varphi \Leftrightarrow y \models \varphi$ whenever $x \approx_j y$. This is the analog of the condition that each formula can only refer to finitely many constants, and thus cannot distinguish objects equivalent with respect to $\approx_j$ for some j.

Representation systems for relational databases We give them for the OWA and CWA semantics. Let $\mathcal{D}(\sigma)$ be the set of all relational databases of vocabulary σ over $\mathsf{Const} \cup \mathsf{Null}$, and $\mathcal{C}(\sigma)$ the set of all such databases without nulls. The database 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 equivalence relations Iso are as explained earlier: they are given by isomorphisms that are the identity on finite sets of constants.

To describe formulae expressing knowledge, we need the notation $\text{PosDiag}(D)$ for the positive diagram of D in the vocabulary including constants for each $a \in \mathsf{Const}$, where with each null $\bot_i$ in D we associate a variable x_i. For instance, if D contains a relation R with tuples $(1, 2), (2, \bot_1), (\bot_1, \bot_2)$, then $\text{PosDiag}(D) = R(1, 2) \wedge R(2, x_1) \wedge R(x_1, x_2)$.

Then the OWA representation system is $\mathbb{RS}_{\text{OWA}}(\sigma) = \langle \mathbb{D}_{\text{OWA}}(\sigma), \exists\mathsf{Pos}, \models\rangle$. For each D with $\mathsf{Null}(D) = \{\bot_1, \ldots, \bot_n\}$, we have $\delta_D = \exists x_1, \ldots, x_n\, \text{PosDiag}(D)$.

For CWA, we need an extension of the class of positive formulae, introduced by [Compton, 1983]. The class, denoted by $\mathsf{Pos}^{\forall \mathsf{G}}$, extends Pos with a special type of guarded formulae. It is defined as the closure of positive atoms of the form $R(\bar{x})$ and $x = y$ under $\wedge, \vee, \forall, \exists$ 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 $\forall \bar{x}\, (R(\bar{x}) \to \varphi(\bar{x}, \bar{y}))$ is a $\mathsf{Pos}^{\forall \mathsf{G}}$ formula.

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

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

Properties of representation systems We now list some of the basic properties of representation systems. For every object x, we have

- $\Box_{\mathcal{O}} [\![x]\!] = x$;

- $\Box_{\mathcal{K}}[\![x]\!] = \delta_x$;
- $\mathsf{Mod}(\delta_x) = \mathsf{Mod}(\mathsf{Th}(x))$.

Let $\approx = \bigcup_{j\in J} \approx_j$, and let $[\![x]\!]_{\approx} = \{c \in [\![x]\!] \mid c \approx x\}$. In case of relational databases, $D \approx D'$ if D, D' are isomorphic objects; for instance, $D = \{(\bot, \bot)\}$ and $D' = \{(1,1)\}$ are isomorphic. Then $\mathsf{Th}([\![x]\!]) = \mathsf{Th}([\![x]\!]_{\approx}) = \mathsf{Th}(x)$ for every object x.

$\Box_{\mathcal{K}}$ as a greatest lower bound There is a well-known preorder on sets of formulae, namely implication: $\Phi \vdash \Psi$ iff $\mathsf{Mod}(\Phi) \subseteq \mathsf{Mod}(\Psi)$. Thus, for any set of formulae Φ, we can look at its greatest lower bound in this preorder, i.e., a formula φ so that $\varphi \vdash \Phi$, and whenever $\psi \vdash \Phi$, we have $\psi \vdash \varphi$. If such a formula exists, it is denoted by $\bigwedge \Phi$. Then:

Theorem 1 *In a representation system, $\Box_{\mathcal{K}} X = \bigwedge \mathsf{Th}(X)$ for every set X of objects.*

5 Defining certain answers to queries

Now we move to answering queries. A query is a mapping Q that takes an object and returns another object. For instance, relational queries take relational databases and return relational databases (most commonly, single relations: queries in FO, or in commercial languages such as SQL, are such).

Thus, for two database domains $\mathbb{D} = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!], \mathrm{Iso}\rangle$ and $\mathbb{D}' = \langle \mathcal{D}', \mathcal{C}', [\![\,]\!]', \mathrm{Iso}'\rangle$, a *query* $Q : \mathbb{D} \to \mathbb{D}'$ is a mapping associating with an object $x \in \mathcal{D}$ its answer, $Q(x) \in \mathcal{D}'$.

The key requirement to queries is the following:

> *if we know more about the input, then we know more about the output.*

By "knowing more", we mean the information orderings $\preceq$ and $\preceq'$, given by the semantics. Indeed, if we have $x \preceq y$ and we want to find $Q(y)$, then as a start, we could have used a less informative object x to compute $Q(x)$. Thus, $Q(y)$ should give us at least the information contained in $Q(x)$. If it does not, it simply means that the semantics $[\![\,]\!]'$ of query answers was chosen incorrectly.

Formally, preserving informativeness means that if $x \preceq y$ then $Q(x) \preceq' Q(y)$. That is, the query Q must be monotone with respect to the information orderings given by the semantics of query inputs and query answers. Note that using blindly some fixed semantics for query results – as in fact is often done – does not necessarily make sense.

Certain answers to Q on an object x represent certain information in the set $Q([\![x]\!]) = \{Q(c) \mid c \in [\![x]\!]\}$. We have seen that there are two ways to define it: as object, and as knowledge. For the letter, we need to have a representation system $\mathbb{RS} = \langle \mathbb{D}', \mathbb{F}, \models\rangle$ over the target domain $\mathbb{D}'$. If we have it, we can either extract the most general object representing $Q([\![x]\!])$, or the most general knowledge representing $\mathsf{Th}(Q([\![x]\!]))$. That is, we have two certain answers notions, as objects and as knowledge:

- As objects: $\mathsf{cert}_{\mathcal{O}}(Q, x) = \Box_{\mathcal{O}} Q([\![x]\!])$;
- As knowledge: $\mathsf{cert}_{\mathcal{K}}(Q, x) = \Box_{\mathcal{K}} Q([\![x]\!])$.

Comparing with relational theory Let us now review the standard approach to query answering in relational databases. Ideally, one tries to find a query answer A so that $[\![A]\!]' = Q([\![D]\!])$. This is often impossible, in fact even for very simple queries [Imielinski and Lipski, 1984]. So the next attempt is to find a formula $\varphi_{Q,D}$ in some logical formalism so that

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

When this happens, one refers to such a logical formalism as a *strong representation system* (see [Abiteboul *et al.*, 1995; Imielinski and Lipski, 1984]), which explains why we used the name 'representation system'.

The problem is that the structure of $Q([\![D]\!])$ may be too "irregular" to be described by a nice formalism. One known example for FO queries under CWA involves very ad hoc formulae that do not correspond to nice syntactic subclasses of FO, see [Imielinski and Lipski, 1984]. If the set $Q([\![D]\!])$ does not happen to be of the form $\mathsf{Mod}(\varphi)$ for some nice formula φ, the approach adopted in the database literature is to consider the object $\bigcap Q([\![D]\!])$ as the answer. This is completely ad hoc, however.

It seems much better to ask then, in place of (2), for an answer $\varphi_{Q,D}$ that is *equivalent to the theory* of $Q([\![D]\!])$, rather than defining $Q([\![D]\!])$ precisely. That is, we replace (2) with

$$\mathsf{Mod}(\varphi_{Q,D}) \;=\; \mathsf{Mod}(\mathsf{Th}(Q([\![D]\!]))) \tag{3}$$

which is, of course, our definition of certain answers expressed as knowledge.

It is easy to see that (2) implies (3). Thus, the notion of certain answers as knowledge in a representation system is a weakening of the notion of the strong representation system, but much less ad hoc that replacing $Q([\![D]\!])$ with $\bigcap Q([\![D]\!])$.

Example: when representation system makes a difference We can easily construct examples of relational queries Q and representation systems so that (2) fails while (3) is easily achieved. Suppose we have a schema with two relations R, S (for simplicity, just sets), and the query $R - S$ (in FO, $R(x) \wedge \neg S(x)$), and assume closed-world semantics. Consider D in which $R = \{1, 2\}$ and $S = \{\bot\}$. Then $Q([\![D]\!]_{\mathrm{CWA}}) = \{\{1\}, \{2\}, \{1, 2\}\}$. Suppose the representation system is $\langle \mathbb{D}(\sigma), \exists\mathsf{Pos}, \models\rangle$. Then there is no α with $\mathsf{Mod}(\alpha) = Q([\![D]\!]_{\mathrm{CWA}})$ but there is one such that $\mathsf{Mod}(\alpha) = \mathsf{Mod}(\mathsf{Th}(Q([\![D]\!]_{\mathrm{CWA}})))$; in fact, the obvious answer $\alpha = A(1) \vee A(2)$ does the job.

6 Certain answers: correctness for free

The intersection-based definition of certain answers (1) led to many complexity issues: lower bounds such as coNP-hard or even undecidable are common for certain answer computation. In the ideal world, we would like to apply a given query Q itself and be sure its result is correct. That is, we would like to have

$$\mathsf{cert}_{\mathcal{O}}(Q, x) \;=\; Q(x). \tag{4}$$

If (4) holds, it is natural to expect that $\mathsf{cert}_{\mathcal{K}}(Q, x) = \delta_{Q(x)}$. While for actual query answering (4) is the important condition, it turns out that we need $\mathsf{cert}_{\mathcal{K}}(Q, x) = \delta_{Q(x)}$ to obtain it.

When $\mathsf{cert}_{\mathcal{O}}(Q, x) = Q(x)$ holds for every x, we say that Q provides *certainty guarantees*. Essentially, the basic query evaluation computes certain answers.

To ensure this, we need an additional condition of *genericity*, standard in the database context [Abiteboul *et al.*, 1995]. In our abstract framework it is expressed as follows: for every j, there is k so that $x \approx_k y$ implies $Q(x) \approx'_j Q(y)$. Essentially, this condition says that queries applied to isomorphic objects return isomorphic objects. For instance, for FO queries that do not refer to constants, it is usually formulated as $D \approx D' \Rightarrow Q(D) \approx Q(D')$. We use a slightly more refined version that, in the case of logically expressed queries, accounts for constants by using multiple equivalence relations. All queries expressed in FO and other logics over the vocabulary of relation symbols and constants are generic in the standard database domains for relational databases.

Theorem 2 *Let $Q : \mathbb{D} \to \mathbb{D}'$ be a generic query that preserves informativeness, and let $\mathbb{RS} = \langle \mathbb{D}', \mathbb{F}, \models \rangle$ be a representation system over the domain of query answers. Then Q provides certainty guarantees.*

Discussion The result says that the new disciplined definition of certain answers provides correctness guarantees for all queries that are generic and preserve informativeness, as long as we have a representation system for query answers. Let us now discuss the importance of these requirements, and how much do they actually impose on the setting.

The existence of a representation system This condition is essential: one can easily find examples where, in the absence of a representation system, correctness guarantees do not hold. On the positive side, for common semantics (e.g., OWA and CWA), representation systems can easily be constructed.

Genericity This is an almost "free" condition: genericity applies to most of the logical formalisms used for querying databases. Among exceptions are formalisms capable of referring to infinitely many constants (e.g., those occurring in data exchange, where it is sometimes necessary to distinguish constants from nulls [Arenas *et al.*, 2009; Fagin, 2007]). But then another condition can be used in place of genericity, namely a substitution property saying that for every formula φ over query answers and every query Q, there is a formula φ_Q over query inputs so that $x \models \varphi_Q$ iff $Q(x) \models \varphi$. Other exceptions arise in formalisms that treat comparisons of nulls differently from comparisons of constants, e.g., those modeling SQL's nulls [Libkin, 2015]. But then it can be shown that $Q(x) \preceq' \mathsf{cert}_{\mathcal{O}}(Q, x)$; in other words, $Q(x)$ provides an efficiently computable *approximation* of certain answers.

Preserving informativeness This is the crucial condition. However natural it is, it was ignored by most of the work on incompleteness which also ignored the task of choosing the right semantics of query answers. In fact most often one just blindly uses some fixed semantics for query inputs and outputs, which is not justified at all. So to get correctness "for free", one has to work after all, and the important work is to understand the right semantics of query answers which ensures the basic principle of preserving informativeness.

When the output semantics is fixed The case considered most often in the database literature is when one assumes the OWA semantics for query answers, since it corresponds to the ordering $\mathsf{false} \preceq \mathsf{true}$ for Boolean values. More generally, the orderings $\preceq_{\text{OWA}}$ and $\preceq_{\text{CWA}}$, i.e., the information orderings given by $[\![\,]\!]_{\text{OWA}}$ and $[\![\,]\!]_{\text{CWA}}$ are defined as follows: $D \preceq_{\text{OWA}} D'$ iff there is a homomorphism $h : D \to D'$ that preserves constants, and $D \preceq_{\text{CWA}} D'$ iff there is a homomorphism $h : D \to D'$ that preserves constants such that $D' = h(D)$, see [Gheerbrant *et al.*, 2014]. Since false is usually modeled as the empty set and true as the set containing the empty tuple, we do have $\mathsf{false} \preceq_{\text{OWA}} \mathsf{true}$.

The proposition below provides correctness guarantees for classes of FO queries over databases interpreted under OWA and CWA, assuming the $\preceq_{\text{OWA}}$ ordering on query answers.

Proposition 1 *Let query answers be ordered by $\preceq_{\text{OWA}}$. Then every $\exists\mathsf{Pos}$ query is monotone under the $\preceq_{\text{OWA}}$ ordering on inputs, and every $\mathsf{Pos}^{\forall\mathsf{G}}$ query is monotone under $\preceq_{\text{CWA}}$.*

Consequently, $\mathsf{cert}_{\mathcal{O}}(Q, D) = Q(D)$ holds for every $\exists\mathsf{Pos}$ query under $[\![\,]\!]_{\text{OWA}}$ and for every $\mathsf{Pos}^{\forall\mathsf{G}}$ query under $[\![\,]\!]_{\text{CWA}}$.

7 Conclusions

We have argued that the standard definition of certain answers in the database literature has a number of deficiencies, and proposed a new approach to handling queries over incomplete databases. Its key features are as follows.

- Certain answers can be defined at two different levels: as (database) objects, or as knowledge we possess about query answers with certainty.
- The proposed framework, that applies to multiple data models, defines both types of certain answers as greatest lower bounds in orderings that capture the level of informativeness. It also leads to a proper definition of representation systems for query answers.
- If the semantics of query answering is chosen properly, then finding certain answers is reduced to query evaluation, at both object and knowledge level. This tells us that with the right choice of semantics, no new tools are needed for computing query answers and one can rely on the standard database query evaluation engine.

Future work. There are several directions to consider. Commercial languages such as SQL use multi-valued logic for reasoning. Evaluation algorithms of similar nature have been explored in the knowledgebase literature [Levesque, 1998], and in fact recently similar procedures have been designed to address the shortcomings of SQL's three-valued logic [Libkin, 2015]. We would like to pursue this direction, also perhaps in connection with finding very efficient approximations for certain answers, as suggested by [Reiter, 1986] and explored for basic features of SQL in [Libkin, 2015].

We also want to apply the framework to non-relational models, particularly semi-structured and XML, for which incompleteness has been studied extensively [Abiteboul *et al.*, 2006; Barceló *et al.*, 2010; Calvanese *et al.*, 1998; David *et*

al., 2010], and beyond, to graph data, where only preliminary results have been established so far [Barceló *et al.*, 2014; Nikolaou and Koubarakis, 2013].

Acknowledgments Work partly supported by EPSRC grant J015377.

References

[Abiteboul *et al.*, 1991] 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.

[Abiteboul *et al.*, 1995] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[Abiteboul *et al.*, 2006] S. Abiteboul, L. Segoufin, and V. Vianu. Representing and querying XML with incomplete information. *ACM Transactions on Database Systems*, 31(1):208–254, 2006.

[Arenas *et al.*, 2009] Marcelo Arenas, Pablo Barceló, and Juan Reutter. Query languages for data exchange: beyond unions of conjunctive queries. In *International Conference on Database Theory (ICDT)*, pages 73–83, 2009.

[Arenas *et al.*, 2014] Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.

[Barceló *et al.*, 2010] Pablo Barceló, Leonid Libkin, Antonella Poggi, and Cristina Sirangelo. XML with incomplete information. *Journal of the ACM*, 58(1), 2010.

[Barceló *et al.*, 2014] Pablo Barceló, Leonid Libkin, and Juan Reutter. Querying regular graph patterns. *J. ACM*, 61(1), 2014.

[Barwise and Feferman, 1985] J. Barwise and S. Feferman, editors. *Model-Theoretic Logics*. Springer Verlag, 1985.

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

[Buneman *et al.*, 1991] Peter Buneman, Achim Jung, and Atsushi Ohori. Using powerdomains to generalize relational databases. *Theoretical Computer Science*, 91(1):23–55, 1991.

[Calvanese *et al.*, 1998] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Semi-structured data with constraints and incomplete information. In *Description Logics*, 1998.

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

[David *et al.*, 2010] Claire David, Leonid Libkin, and Filip Murlak. Certain answers for XML queries. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 191–202, 2010.

[Fagin, 2007] Ronald Fagin. Inverting schema mappings. *ACM Transactions on Database Systems*, 32(4), 2007.

[Gheerbrant *et al.*, 2014] Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. Naïve evaluation of queries over incomplete databases. *ACM Transactions on Database Systems*, 39(4):231, 2014.

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

[Gunter, 1992] Carl Gunter. *Semantics of Programming Languages: Structures and Techniques*. MIT Press, 1992.

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

[Kontchakov *et al.*, 2011] Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. The combined approach to ontology-based data access. In *IJCAI*, pages 2656–2661, 2011.

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

[Lenzerini, 2002] Maurizio Lenzerini. Data integration: a theoretical perspective. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002.

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

[Libkin, 2011] Leonid Libkin. Incomplete information and certain answers in general data models. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 59–70, 2011.

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

[Libkin, 2014b] Leonid Libkin. Incomplete information: what went wrong and how to fix it. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 1–13, 2014.

[Libkin, 2015] Leonid Libkin. SQL's three-valued logic and certain answers. In *18th International Conference on Database Theory (ICDT)*, 2015.

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

[Nikolaou and Koubarakis, 2013] Charalampos Nikolaou and Manolis Koubarakis. Incomplete information in RDF. In *RR*, pages 138–152, 2013.

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

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

[Suciu *et al.*, 2011] D. Suciu, D. Olteanu, C. Re, and C. Koch. *Probabilistic Databases*. Morgan&Claypool Publishers, 2011.