Certain Answers as Objects and Knowledge

Certain Answers as Objects and Knowledge

Leonid Libkin
School of Informatics, 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 counter-intuitive or just plain wrong results, and propose an alternative framework for defining certain answers.

The idea of the framework is to move away from the standard, in the database literature, assumption that query results be given in the form of a database object, and to allow instead two alternative representations of answers: as objects defining all other answers, or as knowledge we can deduce with certainty about all such answers. We show that the latter is often easier to achieve than the former, that in general certain answers need not be defined as intersection, and may well contain missing information in them. We also show that with a proper choice of semantics, we can often reduce computing certain answers – as either objects or knowledge – to standard query evaluation. We describe the framework in the most general way, applicable to a variety of data models, and test it on three concrete relational semantics of incompleteness: open, closed, and weak closed world.

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, Kanellakis, and Grahne 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 the shifting focus in database applications, owing to the ever increasing amounts of data as well as data heterogeneity, 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), and data quality (Fan and Geerts 2012).

The central problem in the database field is of course query answering, and in the presence of incompleteness, one looks for *certain answers*: those that do not depend on the interpretation of unknown data. The concept was first mentioned in (Grant 1977) and formally defined 35 years ago in (Lipski 1979) as follows. Assume that the semantics $[\![D]\!]$ of an incomplete database D is given as the set of all complete databases D' which D can represent. Then the certain answer to Q on D is defined as $\mathsf{cert}_\cap(Q, D) = \bigcap\{Q(D') \mid D' \in [\![D]\!]\}$. Since database queries produce collections (sets, multisets, etc.), it makes sense to talk about their intersection. This definition has been universally applied to all the semantics of incompleteness, and in all the scenarios such as those listed above. 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 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 present here, 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* $Q([\![D]\!]) = \{Q(D') \mid D' \in [\![D]\!]\}$. Such an object A must contain information common to all the objects in $Q([\![D]\!])$: that is, it must be no more informative than any of the objects in $Q([\![D]\!])$.

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 $D' \in [\![D]\!]$? The answer depends on the semantics. Under the common *open-world* semantics, the answer is yes: in fact $(1,2)$ is precisely the greatest lower bound of $Q([\![D]\!])$ under the ordering whose meaning is "being less informative". But under the equally common *closed-world* semantics, the answer is no. Even worse, $(1,2)$ is not less informative than any of the answers $Q(D')$ for $D' \in [\![D]\!]$ 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.

The problem with returning the single tuple $(1,2)$ as the certain answer becomes even more pronounced if we follow 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 is it the only fact that is implied? Of course not: under the open-world semantics, we can deduce $\exists x\, R(1,2) \wedge R(3,x)$ with certainty, adding the fact that there is a tuple whose first component is 3. And under the closed world semantics, we know for certain even more, for instance $\exists x \forall y\ (y = 1 \vee y = 2 \vee y = 3 \vee y = x)$, since we cannot expand the database.

Even from this simple example, we learn a few lessons:

- Certain answers can be presented as both objects and logical formulae;
- they depend on both logical languages and semantics used; and
- taking intersection and removing missing values from the answers is not always the right way to compute them.

Note that viewing answers as formulae brings us closer to knowledge bases. The difference though is that while in databases we start with objects and produce objects, and in querying knowledge bases we start with logical theories and produce logical theories, here we may also combine the two, starting with an object, and returning a formula, or a theory, as the result.

The goal of this paper 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), rather than treat them separately. 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.
- In general, it is easier to find certain answers as knowledge as opposed to certain answers as objects: sometimes the former exists and the latter does not. Thus, representing certain knowledge about query answers is not a mere convenience, it may well be a necessity.

Most of the results in the paper are shown in an abstract setting, for two reasons. First, it makes them applicable to other data models, beyond relational databases (e.g., XML and graph data). 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 three common relational semantics – open world, closed world, and weak closed world – to translate general results into concrete examples, to test the approach.

Organization. After presenting basic facts about relational incompleteness in Section 2, we introduce the abstract setting and notions of certainty as objects and knowledge in Section 3. Section 4 defines representation systems and connects knowledge certainty with greatest lower bounds. Section 5 defines certain answers for queries, and Section 6 shows how to compute them. Summary and future work are in Section 7.

2 Background: relational incompleteness

Incomplete databases These have 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 $\mathrm{adom}(D) = \mathsf{Const}(D) \cup \mathsf{Null}(D)$. A *complete* database D has no nulls, i.e., $\mathrm{adom}(D) \subseteq \mathsf{Const}$.

Semantics and valuations 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, Hull, and Vianu 1995; Imielinski and Lipski 1984). These are the *closed world assumption*, or CWA semantics:

$$[\![D]\!]_{\text{CWA}} = \{v(D) \mid v \text{ is a valuation}\},$$

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

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

We shall also consider the *weak* CWA, or WCWA semantics, inspired by (Reiter 1980), given by

$$[\![D]\!]_{\text{WCWA}} = \left\{ v(D) \cup D' \;\middle|\; \begin{array}{l} v \text{ is a valuation and} \\ \text{adom}(D') \subseteq \text{adom}(v(D)) \end{array} \right\}.$$

That is, under CWA, we simply instantiate nulls by constants; under OWA, we can also add arbitrary tuples, and under WCWA, we can only add tuples formed by elements already present. For instance, if $D_0 = \{(\bot, \bot')\}$, then $[\![D]\!]_{\text{CWA}}$ only has instances $\{(c, c')\}$ for $c, c' \in \text{Const}$, while $[\![D]\!]_{\text{WCWA}}$ can have in addition instances that add to (c, c') tuples (c, c), (c', c'), and (c', c), and $[\![D]\!]_{\text{OWA}}$ has all instances containing at least one tuple.

Information orderings With an arbitrary semantics $[\![\,]\!]$ of incompleteness, we can associate an *information ordering*

$$D \preceq D' \;\Leftrightarrow\; [\![D']\!] \subseteq [\![D]\!].$$

The meaning is that D' is more informative than D. Indeed, the more possible objects we have in the semantics of an incomplete database, the less we know about it. Note that $\preceq$ is a preorder, i.e., it is reflexive and transitive, but both $D \preceq D'$ and $D' \preceq D$ may be true if $[\![D]\!] = [\![D']\!]$.

We write $\preceq_*$ for the ordering given by $[\![\,]\!]_*$, where $*$ is one of CWA, WCWA, or OWA. These can be described in terms of homomorphisms. Given two relational databases D and D', a homomorphism $h : D \to D'$ is a map from $\text{adom}(D)$ to $\text{adom}(D')$ such that $h(c) = c$ for each constant c, and such that for every relation symbol R and a tuple $\bar{t} \in R^D$, the tuple $h(\bar{t})$ is in $R^{D'}$. The image of the homomorphism is denoted by $h(D)$. A homomorphism is *onto* if $\text{adom}(h(D)) = \text{adom}(D')$, and *strong onto* if $D' = h(D)$.

It is known (Libkin 2011; Gheerbrant, Libkin, and Sirangelo 2013) that $D \preceq_* D'$ iff there exists a

- homomorphism $h : D \to D'$, for $* =$OWA;
- onto homomorphism $h : D \to D'$, for $* =$WCWA;
- strong onto homomorphism $h : D \to D'$, for $* =$CWA.

Classical definition of certain answers Given an incomplete database D, a semantics of incompleteness $[\![\,]\!]$, and a query Q, the standard notion of certain answers under $[\![\,]\!]$ is

$$\text{cert}_\cap(Q, D) = \bigcap \{Q(R) \mid R \in [\![D]\!]\}.$$

Note that $\text{cert}_\cap(Q, D)$ cannot contain any tuples with nulls.

In some cases, these answers are 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 $\text{cert}_\cap(Q, D)$ are $\{(1, 2)\}$. In general of course the two need not coincide, as finding certain answers may be computationally very expensive: for instance, undecidable under OWA, or CONP-hard under CWA for first-order queries (Abiteboul, Kanellakis, and Grahne 1991).

Logics 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.

It is known that $\text{cert}_\cap(Q, D) = Q^{\mathsf{C}}(D)$ for $\exists\mathsf{Pos}$ queries under OWA (Imielinski and Lipski 1984), and for Pos queries under WCWA (Gheerbrant, Libkin, and Sirangelo 2013). There is a fragment for which the equality holds under CWA as well, and it will be given later.

3 Objects and knowledge

The key idea, as explained in the introduction, is to decouple objects and their descriptions in terms of some logical formalisms, i.e., to decouple objects and knowledge about them. 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, Libkin, and Sirangelo 2013).

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 incomplete database x.
- We require that for $c \in [\![x]\!]$, two conditions hold: $c \in [\![c]\!]$ and $[\![c]\!] \subseteq [\![x]\!]$.

The last two conditions hold in the standard semantics of incompleteness: they say that a complete object should denote at least itself, and that we have less uncertainty about an object c in the semantics of x than about x itself.

As for relational semantics seen earlier, we have the *information ordering*

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

Note that $c \in [\![x]\!]$ implies $x \preceq c$, i.e., an incomplete object is less informative than the objects it represents.

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 $\mathsf{Mod}(\Phi) = \bigcap_{\varphi \in \Phi} \mathsf{Mod}(\varphi)$.

Note that Mod and Th define a Galois connection between $\mathcal{D}$ and $\mathbb{F}$; in particular $\mathsf{Mod}(\mathsf{Th}(\cdot))$ is a closure operator.

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 $\Box_{\mathcal{O}} X$.

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. Of course 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 $\mathsf{Mod}(\varphi) = \mathsf{Mod}(\mathsf{Th}(X))$. We denote such a formula 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 $\mathsf{Mod}(\Box_{\mathcal{K}} X) = \mathsf{Mod}(\mathsf{Th}(X))$.

We now make a couple observations about these concepts. First, neither $\Box_{\mathcal{O}} X$ nor $\Box_{\mathcal{K}} X$ need exist in general (in fact it is easy to come up with the examples of preorders without greatest lower bounds).

Second, even if they exist, they are not 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.

Also note that a rudimentary case of $\Box_{\mathcal{K}} X$ was used in (David, Libkin, and Murlak 2010), which looked at certain answers for XML queries. There, the set $\mathbb{F}$ of formulae was $\mathcal{D}$ itself, and $x \models y$ simply meant $y \preceq x$. Just in that simple case, Theorem 1 below is a consequence of (Libkin 2011).

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 $\Box_{\mathcal{K}} X = R(1,2)$.

If $X = [\![D]\!]_{\text{CWA}}$, then $\bigwedge X$ is still D, but we need a more complex class to describe $\Box_{\mathcal{K}} X$; in fact the two formulae above can easily be shown to violate the definition of $\Box_{\mathcal{K}} X$ in this case. We shall continue with this example in the next section and present a formula $\Box_{\mathcal{K}} X$ under CWA, that comes from a more expressive subclass of FO.

4 Representation systems

So far we imposed no conditions on pre-representation systems. We now define representation systems, which are pre-representation systems with two conditions imposed. These conditions, that hold in the standard semantics of incompleteness, say essentially that the sets of objects and formulae are not too "thin": there are enough complete objects, and there are formulae defining some fairly 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 $\approx_k \subseteq \approx_j \cap \approx_{j'}$.

Coming back to the relational intuition, $j \in J$ enumerate finite sets of constants $C_j \subset \mathsf{Const}$, and $D \approx_j D'$ means that there is an isomorphism between D and D' preserving constants in C_j. Then 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 First, we assume that formulae are closed under conjunction, i.e. for $\varphi, \psi \in \mathbb{F}$ we have $\varphi \wedge \psi \in \mathbb{F}$ with $\mathsf{Mod}(\varphi \wedge \psi) = \mathsf{Mod}(\varphi) \cap \mathsf{Mod}(\psi)$.

The second assumption is that the set $\uparrow x = \{y \mid x \preceq y\}$ of objects more informative than x can be defined by a formula. We shall see shortly that this is true in the classical incompleteness semantics, and that this is equivalent to finite axiomatizability of $\mathsf{Th}(x)$.

Formally the condition states that for each object x, there is a formula δ_x in $\mathbb{F}$, such that $\mathsf{Mod}(\delta_x) = \uparrow x$.

Representation systems A *representation system* is a triple $\mathbb{RS} = \langle \mathbb{D}, \mathbb{F}, \models \rangle$, where:

- $\mathbb{D}$ is a database domain and $\langle \mathbb{D}, \mathbb{F}, \models \rangle$ is a pre-representation system whose formulae are closed under conjunction;
- for each $x \in \mathcal{D}$, there is a formula δ_x with $\mathsf{Mod}(\delta_x) = \uparrow x$;
- for each $\varphi \in \mathbb{F}$, there is $j \in J$ so that $x \models \varphi \Leftrightarrow y \models \varphi$ whenever $x \approx_j y$.

The last condition is essentially the analog of the condition that a formula can only refer to finitely many constants, and thus cannot distinguish objects equivalent with respect to $\approx_j$ for some j.

Examples of domains and representation systems

We now provide examples of representation systems corresponding to relational OWA, WCWA, and CWA semantics. We use the notation $\mathcal{D}(\sigma)$ for the set of all relational databases of vocabulary σ over $\mathsf{Const} \cup \mathsf{Null}$, and $\mathcal{C}(\sigma)$ for the set of all such databases that do not use nulls in Null. The database domains will be of the form $\mathbb{D}_*(\sigma) = \langle \mathcal{D}(\sigma), \mathcal{C}(\sigma), [\![\,]\!]_*, \text{Iso} \rangle$, where $*$ is one of OWA, WCWA, or CWA.

The equivalence relations Iso are as follows. Let J enumerate all the finite subsets of Const. Given such a finite set C, we have $D \approx_C D'$ if there is an isomorphism $f : D \to D'$ such that both f and f^{-1} are identity on C.

In what follows, we describe formulae $\mathbb{F}$ and formulae δ_D for each D. Let $\text{PosDiag}(D)$ be 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 relation R with tuples $(1, 2), (2, \bot_1), (\bot_1, \bot_2)$, then $\text{PosDiag}(D) = R(1, 2) \wedge R(2, x_1) \wedge R(x_1, x_2)$.

OWA 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 $\delta_D = \exists x_1, \ldots, x_n\, \text{PosDiag}(D)$.

WCWA The WCWA representation system is $\mathbb{RS}_{\text{WCWA}}(\sigma) = \langle \mathbb{D}_{\text{WCWA}}(\sigma), \mathsf{Pos}, \models \rangle$. For each D with $\mathsf{Const}(D) = \{a_1, \ldots, a_m\}$ and $\mathsf{Null}(D) = \{\bot_1, \ldots, \bot_n\}$, the formula δ_D is

$$\exists x_1 \ldots x_n \big(\text{PosDiag}(D) \wedge \forall y \, (\bigvee_{i=1}^{m} y = a_i \ \vee \ \bigvee_{i=1}^{n} y = x_i)\big)$$

CWA We need to define an extension of the class of positive formulae, introduced by (Compton 1983) and used recently in (Gheerbrant, Libkin, and Sirangelo 2013). 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

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

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

Proposition 1 *Each of* $\mathbb{RS}_{\text{OWA}}(\sigma)$*,* $\mathbb{RS}_{\text{WCWA}}(\sigma)$*, and* $\mathbb{RS}_{\text{CWA}}(\sigma)$ *is a representation system.*

Proof sketch. Conditions on Iso are easily checked: sets $[\![D]\!]_C$ are nonempty, since one can replace nulls with distinct constants from $\mathsf{Const} - C$, and $\approx_{C \cup C'} \subseteq \approx_C \cap \approx_{C'}$. Monotonicity follows from the description of orderings $\preceq_*$ by the existence of homomorphisms (see Section 2) and the fact that $\exists\mathsf{Pos}$ (resp., Pos and $\mathsf{Pos}^{\forall \mathsf{G}}$) formulae are preserved under homomorphisms (resp., onto and strong onto), see (Rossman 2008; Chang and Keisler 1990; Compton 1983). And the properties of δ_D again follow from the properties of the ordering, as one can easily check that $\mathsf{Mod}(\delta_D)$ are the structures into which there is a homomorphism (resp., onto or strong onto homomorphism) from D. ☐

Properties of representation systems

We now collect some useful properties of representation systems. First, we look at certain information contained in sets $[\![x]\!]$ and prove that, as expected, it is represented at the object level by x, and at the knowledge level by δ_x.

Proposition 2 *In a pre-domain,* $\Box_{\mathcal{O}}[\![x]\!] = x$ *for every* x*.*

Proof. Assume that $x \neq \bigwedge [\![x]\!]$. Then there is $y \not\preceq x$ such that $y \preceq c$ for each $c \in [\![x]\!]$. Since $y \not\preceq x$ we have $[\![x]\!] \not\subseteq [\![y]\!]$, i.e., there is $c \in [\![x]\!]$ such that $c \notin [\![y]\!]$. But since $y \preceq c$, we have $[\![c]\!] \subseteq [\![y]\!]$, and since $c \in \mathcal{C}$, this implies $c \in [\![y]\!]$ as $c \in [\![c]\!]$, which gives us the desired contradiction. ☐

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. Note that FO formulae not using constants are preserved by $\approx$. In general though, objects related by $\approx_j$ may not agree on all the formulae of $\mathbb{F}$ (e.g., D and D' do not agree on $\exists x\ (x = 1)$) and hence there are potentially formulae in $\mathsf{Th}([\![x]\!]_{\approx_j})$ which are not satisfied by x. However,

Proposition 3 *In a representation system,* $\mathsf{Th}([\![x]\!]) = \mathsf{Th}([\![x]\!]_{\approx}) = \mathsf{Th}(x)$ *for every* x*.*

Proof. It suffices to prove $\mathsf{Th}([\![x]\!]_{\approx}) = \mathsf{Th}(x)$ since $\mathsf{Th}(x) \subseteq \mathsf{Th}([\![x]\!]) \subseteq \mathsf{Th}([\![x]\!]_{\approx})$. Let $c \in [\![x]\!]_{\approx}$; then $x \preceq c$ and thus $\mathsf{Th}(x) \subseteq \mathsf{Th}(c)$, and hence $\mathsf{Th}(x) \subseteq \mathsf{Th}([\![x]\!]_{\approx})$. Conversely, take $\varphi \in \mathsf{Th}([\![x]\!]_{\approx})$. We know that there is $j \in J$ so that whenever $y \approx_j y'$, then y, y' agree on φ. Note that $\varphi \in \mathsf{Th}([\![x]\!]_{\approx_j})$, since $[\![x]\!]_{\approx_j} \subseteq [\![x]\!]_{\approx}$. We know that $[\![x]\!]_{\approx_j} \neq \emptyset$, so pick $c \in [\![x]\!]_{\approx_j}$. Since $\varphi \in \mathsf{Th}([\![x]\!]_{\approx_j})$, we have $c \models \varphi$, and since $c \approx_j x$, we have $x \models \varphi$. Hence $\varphi \in \mathsf{Th}(x)$, as required. □

Now we can show:

Proposition 4 *In a representation system, $\delta_x = \Box_{\mathcal{K}}[\![x]\!]$ for every x.*

Proof. We first show that $\mathsf{Mod}(\mathsf{Th}(x)) = \uparrow x$. Let $y \succeq x$. By monotonicity we have $y \in \mathsf{Mod}(\mathsf{Th}(x))$. Conversely let $y \not\succeq x$; then $\delta_x \in \mathsf{Th}(x) - \mathsf{Th}(y)$ and hence $y \notin \mathsf{Mod}(\mathsf{Th}(x))$. Thus indeed $\mathsf{Mod}(\mathsf{Th}(x)) = \uparrow x$. Since $\uparrow x = \mathsf{Mod}(\delta_x)$ and $\mathsf{Mod}(\mathsf{Th}(x)) = \mathsf{Mod}(\mathsf{Th}([\![x]\!]))$ by Proposition 3, we have $\mathsf{Mod}(\delta_x) = \mathsf{Mod}(\mathsf{Th}([\![x]\!]))$, i.e., $\delta_x = \Box_{\mathcal{K}}[\![x]\!]$. □

Corollary 1 *In a representation system:*

- $\mathsf{Mod}(\delta_x) = \mathsf{Mod}(\mathsf{Th}(x))$, *and*
- $\Box_{\mathcal{O}}[\![x]\!] \models \Box_{\mathcal{K}}[\![x]\!]$

for every x.

We finally give a condition equivalent to the existence of formulae δ_x. A set Φ of formulae is *finitely axiomatizable* if there is a finite set Φ_0 such that $\mathsf{Mod}(\Phi) = \mathsf{Mod}(\Phi_0)$.

Proposition 5 *In a pre-representation system $\mathbb{RS}^\circ$ whose formulae are closed under conjunction, the following are equivalent:*

1. *Each $\mathsf{Th}(x)$ is finitely axiomatizable, and $\mathsf{Th}(x) \subseteq \mathsf{Th}(y)$ implies $x \preceq y$ for all x, y;*
2. *for each x, there is a formula δ_x so that $\mathsf{Mod}(\delta_x) = \uparrow x$.*

Proof sketch. If Φ is a finite axiomatization of $\mathsf{Th}(x)$, it is easy to see that δ_x can be taken to be $\bigwedge\{\varphi \mid \varphi \in \Phi\}$. Conversely, one shows that δ_x axiomatizes $\mathsf{Th}(x)$. □

$\Box_{\mathcal{K}}$ as a greatest lower bound

We now show that $\Box_{\mathcal{K}} X$ can be viewed as a greatest lower bound as well. Note that we have 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 $\bigwedge \Phi$. Note that as with objects, $\vdash$ is a preorder, so technically $\bigwedge \Phi$ is a set of formulae, all of which, however, are equivalent, so we shall write $\varphi = \bigwedge \Phi$ when φ is one of such formulae.

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

Proof. Assume that $\alpha = \bigwedge \mathsf{Th}(X)$ exists. We have $\mathsf{Mod}(\alpha) \subseteq \mathsf{Mod}(\varphi)$ for each $\varphi \in \mathsf{Th}(X)$ and thus $\mathsf{Mod}(\alpha) \subseteq \mathsf{Mod}(\mathsf{Th}(X))$. Suppose, for the sake of contradiction, that $\mathsf{Mod}(\alpha) \subsetneq \mathsf{Mod}(\mathsf{Th}(X))$, and take $y \in \mathsf{Mod}(\mathsf{Th}(X)) - \mathsf{Mod}(\alpha)$. Since $y \models \varphi$ for each $\varphi \in \mathsf{Th}(X)$, we have that the same is true for every $z \succeq y$, and hence $\mathsf{Mod}(\delta_y) \subseteq \mathsf{Mod}(\varphi)$ for each $\varphi \in \mathsf{Th}(X)$. Thus $\delta_y \vdash \mathsf{Th}(X)$, and by the definition of α as the greatest lower bound, we have $\delta_y \vdash \alpha$, and thus $\mathsf{Mod}(\delta_y) \subseteq \mathsf{Mod}(\alpha)$.

Now assume $\mathsf{Mod}(\alpha) \subseteq \mathsf{Mod}(\delta_y)$. Then $\mathsf{Mod}(\alpha) = \mathsf{Mod}(\delta_y)$ and $y \in \mathsf{Mod}(\alpha)$, a contradiction. Hence, $\mathsf{Mod}(\alpha) \not\subseteq \mathsf{Mod}(\delta_y)$. But then $\mathsf{Mod}(\delta_y) \not\subseteq \mathsf{Mod}(\alpha)$ since $y \not\models \alpha$, and at the same time $\mathsf{Mod}(\delta_y) \subseteq \mathsf{Mod}(\mathsf{Th}(X))$ since $y \in \mathsf{Mod}(\mathsf{Th}(X))$. Thus sets $\mathsf{Mod}(\delta_y)$ and $\mathsf{Mod}(\alpha)$ are incomparable subsets of $\mathsf{Mod}(\mathsf{Th}(X))$: in particular, $\alpha \vdash \mathsf{Th}(X)$, $\delta_y \vdash \mathsf{Th}(X)$, and yet neither $\alpha \vdash \delta_y$ nor $\delta_y \vdash \alpha$ holds, contradicting the assumption that $\alpha = \bigwedge \mathsf{Th}(X)$. This shows that $\mathsf{Mod}(\mathsf{Th}(X)) - \mathsf{Mod}(\alpha) = \emptyset$ and thus $\mathsf{Mod}(\alpha) = \mathsf{Mod}(\mathsf{Th}(X))$.

Conversely, suppose we have α so that $\mathsf{Mod}(\alpha) = \mathsf{Mod}(\mathsf{Th}(X)) = \bigcap_{\varphi \in \mathsf{Th}(X)} \mathsf{Mod}(\varphi)$. Then $\mathsf{Mod}(\alpha) \subseteq \mathsf{Mod}(\varphi)$ for each $\varphi \in \mathsf{Th}(X)$ and thus $\alpha \vdash \varphi$ for each such φ. If there is any other formula ψ such that $\psi \vdash \varphi$ for each $\varphi \in \mathsf{Th}(X)$, then $\mathsf{Mod}(\psi) \subseteq \bigcap_{\varphi \in \mathsf{Th}(X)} \mathsf{Mod}(\varphi) = \mathsf{Mod}(\alpha)$ and thus $\psi \vdash \alpha$, proving that $\alpha = \bigwedge \mathsf{Th}(X)$. □

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, $Q(x) \in \mathcal{D}'$.

We require that if $x \preceq y$ then $Q(x) \preceq' Q(y)$, where $\preceq'$ is the ordering associated with the semantics $[\![\,]\!]'$. Indeed, the semantics of answers must be such that if we know more about the input, then we know more about the output. In particular, using blindly the same semantics for both databases and query results (as is often actually 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

$$\mathsf{Mod}(\varphi_{Q,D}) = Q([\![D]\!]) \qquad (1)$$

When this happens, one refers to such a logical formalism as a *strong representation system* (see (Abiteboul, Hull, and Vianu 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. For instance, it is known that under CWA, for relational calculus queries formulae $\varphi_{Q,D}$ can be of the following form: $\exists \bar{u}\ (\alpha(\bar{u}) \wedge \bigwedge_{R\in\sigma} \forall \bar{x}\ (R(\bar{x}) \leftrightarrow \bigvee_i (\bar{x} = \bar{v}_i \wedge \beta_i(\bar{x},\bar{u}))))$, where α and β_i are boolean combinations of equalities, $\bar{v}_i$s combine variables from $\bar{u}$ and constants, and $\bar{u}$ ranges over the underlying domain of constants rather than the active domain, see (Imielinski and Lipski 1984). Syntactically, this is quite heavy, and it works only under the CWA.

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: as we have seen in the Introduction (and as we shall see in the next section), in general it does not have much in common with certain information contained in $Q([\![D]\!])$.

It seems much better to ask then, in place of (1), 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 (1) with

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

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

Note that (1) implies (2): this is an immediate consequence of the fact that $\mathsf{Mod}(\cdot)$ and $\mathsf{Th}(\cdot)$ define a Galois connection. 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 (1) fails while (2) 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 $Q([\![D]\!]_{\text{CWA}}) = \{\{1\},\{2\},\{1,2\}\}$. Suppose the representation system is $\langle \mathbb{D}(\sigma), \exists\mathsf{Pos}, \models\rangle$. Then there is no α with $\mathsf{Mod}(\alpha) = Q([\![D]\!]_{\text{CWA}})$ but there is one such that $\mathsf{Mod}(\alpha) = \mathsf{Mod}(\mathsf{Th}(Q([\![D]\!]_{\text{CWA}})))$; in fact, the obvious answer $\alpha = A(1) \vee A(2)$ does the job.

6 Computing certain answers

We now look at computing certain answers. We show that with the proper semantics of query answering, where more informative inputs lead to more informative answers, finding certain answers is reduced to query evaluation:

(a) for objects, $\mathsf{cert}_{\mathcal{O}}(Q,x) = Q(x)$;

(b) for knowledge, $\mathsf{cert}_{\mathcal{K}}(Q,x) = \delta_{Q(x)}$.

Crucially, (a) is a corollary of (b): without a representation system for query answers, (a) may not be true. We explain how these general results apply to relational OWA, WCWA, and CWA semantics. We also revisit the intersection-based definition of certain answers and show that it only makes sense with restricted representation systems under OWA, and is just plain wrong under CWA. Finally, we show that the knowledge approach to certain answers gives us extra flexibility compared to the object approach.

Recall that queries are required to be monotone: $x \preceq y$ implies $Q(x) \preceq' Q(y)$. We need an additional condition of *genericity*, standard in the database context (see, e.g., (Abiteboul, Hull, and Vianu 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 monotone generic query, and let $\mathbb{RS} = \langle \mathbb{D}', \mathbb{F}, \models\rangle$ be a representation system. Then, for every object x,*

- $\mathsf{cert}_{\mathcal{O}}(Q,x) = Q(x)$*, and*
- $\mathsf{cert}_{\mathcal{K}}(Q,x) = \delta_{Q(x)}$*.*

Proof. We start by showing $\mathsf{cert}_{\mathcal{K}}(Q,x) = \delta_{Q(x)}$. By the definition of $\mathsf{cert}_{\mathcal{K}}(Q,x)$ as $\Box_{\mathcal{K}} Q([\![x]\!])$, it suffices to show that

$$\mathsf{Mod}(\delta_{Q(x)}) \ = \ \mathsf{Mod}(\mathsf{Th}(Q([\![x]\!]))) \tag{3}$$

for every x. Since we know that $\mathsf{Mod}(\delta_z) = \mathsf{Mod}(\mathsf{Th}(z))$, for every z, by Corollary 1, we just need to show

$$\mathsf{Th}(Q([\![x]\!])) \ = \ \mathsf{Th}(Q(x)) \tag{4}$$

to conclude (3). Suppose $\varphi \in \mathsf{Th}(Q(x))$. Since $c \succeq x$ for every $c \in [\![x]\!]$, then $Q(x) \preceq' Q(c)$ and $Q(c)$ satisfies φ as well by the properties of representation systems, proving that $\varphi \in \mathsf{Th}(Q([\![x]\!]))$.

Conversely, if $\varphi \in \mathsf{Th}(Q([\![x]\!]))$, consider j such that $z \approx'_j z'$ implies that z, z' agree on φ for every z, z' (which exists by the definition of representation systems). By genericity, we have k so that $y \approx_k y'$ implies $Q(y) \approx'_j Q(y')$ for every y, y'. We know that $[\![x]\!]_{\approx_k}$ is nonempty; this pick an element c in this set. We have $c \in [\![x]\!]$ and $c \approx_k x$. Hence (a) $Q(c) \in Q([\![x]\!])$ and (b) $Q(c) \approx'_j Q(x)$. Then (a) implies that $Q(c) \models \varphi$, and (b) then implies that $Q(x) \models \varphi$, thus showing that $\varphi \in \mathsf{Th}(Q(x))$ and proving (4) and (3).

Now we prove the result about certain answers at the object level. Since Q is monotone, we have $Q(x) \preceq' z$ for each $z \in Q([\![x]\!])$ (we denote this by $Q(x) \preceq' Q([\![x]\!])$. Suppose we have $y \preceq' Q([\![x]\!])$. Then $\mathsf{Th}(y) \subseteq \mathsf{Th}(Q([\![x]\!]))$ and by (4), $\mathsf{Th}(y) \subseteq \mathsf{Th}(Q(x))$. Since $\delta_y \in \mathsf{Th}(y)$, we have $\delta_y \in \mathsf{Th}(Q(x))$, and thus $Q(x) \models \delta_y$ and $y \preceq' Q(x)$. Thus, $Q(x)$ is the greatest lower bound of $Q([\![x]\!])$, i.e., $Q(x) = \mathsf{cert}_{\mathcal{O}}(Q,x)$. □

Remark It is easy to construct examples of monotone queries for which, in the absence of a representation system over query answers, $\mathsf{cert}_{\mathcal{O}}(Q, x) \neq Q(x)$. Thus, it is essential to go via certain answer as knowledge to get its object representation as well.

The following easy observation shows that classes of queries for which certain answers can be computed by directly evaluating the query are closed under composition, giving us a way of producing more complex queries that behave well with respect to certain answer computation.

Proposition 6 *If $Q : \mathbb{D} \to \mathbb{D}'$ and $Q' : \mathbb{D}' \to \mathbb{D}''$ are monotone and generic, then so is $Q' \circ Q : \mathbb{D} \to \mathbb{D}''$.*

Genericity applies to most of the logical formalisms used for querying databases, but not all: formalisms capable of referring to infinitely many constants would not fall into that category. Examples of such formalisms occur, for instance, in data exchange, where one deals with logics that use the predicate $const(\cdot)$ to distinguish constants from nulls, see (Arenas, Barceló, and Reutter 2009; Fagin 2007).

There is another condition we can use in place of genericity, namely a substitution property with respect to two representation systems $\mathbb{RS} = \langle \mathbb{D}, \mathbb{F}, \models \rangle$ and $\mathbb{RS}' = \langle \mathbb{D}', \mathbb{F}', \models' \rangle$. We say that a query Q has the *substitution property* with respect to $\mathbb{RS}$ and $\mathbb{RS}'$ if for each $\varphi \in \mathbb{F}'$, there exists $\varphi_Q \in \mathbb{F}$ so that $x \models \varphi_Q$ iff $Q(x) \models \varphi$. Intuitively, we can think of both φ and Q given as logical formulae, and allow Q to be substituted for predicate symbols used in φ.

Theorem 3 *Let $Q : \mathbb{D} \to \mathbb{D}'$ be a monotone query, and assume we have representation systems $\mathbb{RS}$ and $\mathbb{RS}'$ so that Q has the substitution property with respect to them. Then, for every object x,*

- $\mathsf{cert}_{\mathcal{O}}(Q, x) = Q(x)$*, and*
- $\mathsf{cert}_{\mathcal{K}}(Q, x) = \delta_{Q(x)}$.

Proof. We just need to prove (4) and then the proof of Theorem 2 will apply. The $\mathsf{Th}(Q(x)) \subseteq \mathsf{Th}(Q([\![x]\!]))$ inclusion is proved as before. For the converse, let φ be in $\mathsf{Th}(Q([\![x]\!]))$; take φ_Q that exists by the substitution property. Since $Q(c) \models \varphi$ for each $c \in [\![x]\!]$, we have $c \models \varphi_Q$, and thus $\varphi_Q \in \mathsf{Th}([\![x]\!])$. But we know by Proposition 3 that $\mathsf{Th}([\![x]\!]) = \mathsf{Th}(x)$ and hence $\varphi_Q \in \mathsf{Th}(x)$. But now the substitution property and $x \models \varphi_Q$ imply $Q(x) \models \varphi$, showing $\varphi \in \mathsf{Th}(Q(x))$ and proving the reverse inclusion. ☐

Certain answers for relational queries

We now look at examples of concrete domains and representation systems for relational databases. The domains are always $\mathbb{D}_*(\sigma)$, containing databases of vocabulary σ, with $*$ being the semantics. If a query Q returns sets of m-ary tuples, the domain $\mathbb{D}'$ will have as objects databases of vocabulary σ_m that contains a single m-ary relation $A(\cdot)$. But what will the semantics be? A natural thing to do is to assume the same semantics for query answers and input data. But will this guarantee conditions that let us apply results of the previous section?

It turns out that for classes of queries given by the same first-order formulae we use in representation systems $\mathbb{RS}_*$, we have all the required conditions.

Proposition 7 *Consider an m-ary relational query $Q : \mathbb{D}_*(\sigma) \to \mathbb{D}_*(\sigma_m)$. Then Q is monotone*

- *for $*$ =*OWA*: when Q is an $\exists\mathsf{Pos}$ query;*
- *for $*$ =*WCWA*: when Q is an Pos query;*
- *for $*$ =*CWA*: when Q is an $\mathsf{Pos}^{\forall\mathsf{G}}$ query.*

Moreover, for $*$ being OWA *or* WCWA*, each such Q has the substitution property with respect to $\mathbb{RS}_*(\sigma)$ and $\mathbb{RS}_*(\sigma_m)$.*

Proof. It is very similar to the proof of Proposition 1, as it follows from the preservation properties of the classes of formulae we use. The second property is by a simple inspection of the syntactic structure of the formulae. ☐

Corollary 2 *Let Q be an m-ary relational query. Under the* OWA*,* WCWA*, and* CWA *semantics, if Q comes from $\exists\mathsf{Pos}$, Pos, or $\mathsf{Pos}^{\forall\mathsf{G}}$ respectively, then $\mathsf{cert}_{\mathcal{O}}(Q, D) = Q(D)$ and $\mathsf{cert}_{\mathcal{K}}(Q, D) = \delta_{Q(D)}$ for every database D (with δ coming from $\mathbb{RS}_*(\sigma_m)$, when $*$ is* OWA*,* WCWA*, or* CWA*).*

Returning to the first example from the introduction, assume that our query just returns a relation R itself, and $R = \{(1, 2), (3, \bot)\}$. Then, at the object level, $\mathsf{cert}_{\mathcal{O}}(Q, R) = R$ for all three semantics. Certain answers as knowledge, or $\mathsf{cert}_{\mathcal{K}}(Q, R)$ are given:

- under OWA, by the $\exists\mathsf{Pos}$ formula $\exists x\ \big(R(1, 2) \wedge R(3, x)\big)$;
- under WCWA, by the Pos formula

$$\exists x \left(R(1,2) \wedge R(3,x)\ \wedge\ \forall y \left(\begin{array}{c} y = 1 \vee y = 2 \\ \vee\ \ y = 3 \vee y = x \end{array} \right) \right);$$

- and under CWA, by the $\mathsf{Pos}^{\forall\mathsf{G}}$ formula

$$\exists x \left(\begin{array}{l} \qquad R(1,2) \wedge R(3,x) \\ \wedge\ \ \forall y, z \left(R(y,z) \to \begin{array}{c} (y = 1 \wedge z = 2) \\ \vee\ \ (y = 3 \wedge z = x) \end{array} \right) \end{array} \right)$$

No intersection for certain answers

Now we return to the "classical" of defining certain answers which involves throwing away tuples with nulls and computing intersection, and give more evidence to the fact that this practice does not really work, outside a fairly restrictive representation system under the OWA. As already mentioned in the preliminaries section, for the classes of queries listed in Proposition 7, the well-known equivalences say that $\mathsf{cert}_{\cap}(Q, D)$ is the same $Q^{\mathsf{C}}(D)$, which is $Q(D)$ from which tuples of nulls have been eliminated. We can also view this as a composition of two mappings: Q^{C} is $\pi^{\mathsf{C}} \circ Q$, where π^{C} simply keeps all tuples that only use constants.

The condition that we need on π^{C}, and on Q^{C}, is monotonicity. However, these are not guaranteed in general.

Proposition 8 *The mapping π^{C} is monotone with respect to the ordering $\preceq_{\text{OWA}}$ but it is* not *monotone with respect to $\preceq_{\text{CWA}}$ and $\preceq_{\text{WCWA}}$.*

In fact there are $\exists\mathsf{Pos}$ queries Q such that Q^{C} is not monotone with respect to $\preceq_{\text{CWA}}$ and $\preceq_{\text{WCWA}}$.

Proof. Since there is the identity homomorphism from any substructure to the structure, we have monotonicity under $\preceq_{\text{OWA}}$. For closed world semantics, simply take $R = \{(1,2),(2,\bot)\}$. Then $\pi^{\mathsf{C}}(R) = R^{\mathsf{C}} = \{(1,2)\}$ and $R \not\preceq_{\text{CWA}} R^{\mathsf{C}}$ nor $R \not\preceq_{\text{WCWA}} R^{\mathsf{C}}$. Thus the $\exists\mathsf{Pos}$ query can be taken to be the query returning relation R. □

The consequences of this under WCWA and CWA are twofold. We may easily have situations such that:

- $Q^{\mathsf{C}}(D) \not\preceq \mathsf{cert}_{\mathcal{O}}(Q,D)$, and
- $\mathsf{cert}_{\mathcal{K}}(Q,D)$ does not logically imply $\delta_{Q^{\mathsf{C}}(D)}$.

In other words, such constant 'certain answer' need not represent certain information in $Q(\llbracket D \rrbracket)$, and thus may contain some false positives. This is also true for certain answers defined as intersection because in the example used in Proposition 8, we have $Q^{\mathsf{C}}(D) = \bigcap\{Q(D') \mid D' \in \llbracket D \rrbracket\}$ under both WCWA and CWA semantics.

Thus, the use of the intersection operator is indeed very problematic under WCWA and CWA. In fact this should not come as a surprise at all, even if we deal with query answering at the object level. The $\bigcap$ operator is a greatest lower bound for the $\subseteq$ ordering, and $\subseteq$ results in a more informative instance only under OWA, but not under closed-world assumptions! Thus, choosing $\bigcap$ to define certain answers for all semantics was a completely ad hoc choice.

OWA: when intersection still makes sense We now look at OWA, the only case where the mapping π^{C} is monotone, and where intersection corresponds to the greatest lower bound in an ordering compatible with the semantics.

Given a database domain $\mathbb{D} = \langle \mathcal{D}, \mathcal{C}, \llbracket\,\rrbracket, \text{Iso}\rangle$, let $\mathbb{D}^{\mathsf{C}}$ be its restriction to complete objects, i.e., $\mathbb{D}^{\mathsf{C}} = \langle \mathcal{C}, \mathcal{C}, \llbracket\,\rrbracket, \text{Iso}\rangle$. It is easy to check that this is still a database domain.

Let $\mathsf{PBCatom}$ stand for the set of Positive Boolean Combinations of atoms $R(\bar{c})$, with $\bar{c}$ consisting only of elements of Const. Recall that for OWA, we used the representation system $\mathbb{RS}_{\text{OWA}}(\sigma) = \langle \mathbb{D}(\sigma), \exists\mathsf{Pos}, \models\rangle$. Define a new representation system $\mathbb{RS}^{\mathsf{C}}_{\text{OWA}}(\sigma) = \langle \mathbb{D}^{\mathsf{C}}(\sigma), \mathsf{PBCatom}, \models\rangle$.

Proposition 9 *$\mathbb{RS}^{\mathsf{C}}_{\text{OWA}}(\sigma)$ is a representation system, and if Q is an $\exists\mathsf{Pos}$ query, then Q^{C} is monotone and generic with respect to it.*

Proof. Immediate from the fact that PosDiag(D) for a complete D is a conjunction of atoms of the form $R(\bar{c})$, and that $\exists\mathsf{Pos}$ queries are preserved under homomorphisms. □

Thus, for every database D, and every $\exists\mathsf{Pos}$ query Q, we have $\mathsf{cert}_{\mathcal{O}}(Q^{\mathsf{C}}, D) = Q^{\mathsf{C}}(D)$ and $\mathsf{cert}_{\mathcal{K}}(Q^{\mathsf{C}}, D) = \delta_{Q^{\mathsf{C}}(D)}$ with respect to $\mathbb{RS}^{\mathsf{C}}_{\text{OWA}}(\sigma)$. Hence, under OWA, the traditional way of finding certain answers makes sense if we use a representation system that only 'cares' about complete databases.

Certain knowledge is more flexible

Note that for sets X of the form $Q(\llbracket x \rrbracket)$ for monotone generic queries Q, we have $\Box_{\mathcal{O}} X \models \Box_{\mathcal{K}} X$, as is implied by Theorem 2. In general, however, this need not be the case. The reason is that taking the greatest lower bound of X may lose more information than taking the greatest lower bound of $\mathsf{Th}(X)$, which indicates that working with certain answers as knowledge may be preferable, as they convey more information.

Proposition 10 *There is a representation system and a set of objects X such that $\Box_{\mathcal{O}} X \not\models \Box_{\mathcal{K}} X$.*

Proof. Consider a domain $\mathcal{D}$ with the ordering $x_1 \succeq x_2 \succeq \ldots \succeq x_n \succeq \ldots \succeq x_*$. We assume that all $\approx_j$s are the same, and for each x_i, there is $c_i \succeq x_i$ with $x_i \approx c_i$. The formulas are φ_i for $i \geq 0$. Let $\mathsf{Th}(x_i) = \{\varphi_0\} \cup \{\varphi_j \mid j \geq i\}$ (with $\mathsf{Th}(c_i) = \mathsf{Th}(x_i)$) and $\mathsf{Th}(x_*) = \emptyset$. Let $X = \{x_i \mid i \neq 0\}$. Then $\bigwedge X = x_*$ and $\mathsf{Th}(X) = \{\varphi_0\}$; hence $\Box_{\mathcal{O}} X = x_*$ and $\Box_{\mathcal{K}} X = \varphi_0$ and thus $\Box_{\mathcal{O}} X \not\models \Box_{\mathcal{K}} X$. □

We finish with examples showing that certain answer may not exist as an object while it still exists as knowledge over an appropriate representation system. Take a query $Q = R - S$, and a database D with $R^D = \{1,2\}$ and $S^D = \{3,\bot\}$.

Under CWA, we have $Q(\llbracket D \rrbracket_{\text{CWA}}) = \{\ \{1,2\}, \{1\}, \{2\}\ \}$. If we take the greatest lower bound of these, then for $\preceq_{\text{CWA}}$ (and $\preceq_{\text{WCWA}}$) it simply does not exist. For the subset ordering, or the $\preceq_{\text{OWA}}$ ordering, the greatest lower bound is $\emptyset$, which is, incidentally, what a commercial DBMS would return if one runs this query in SQL (e.g., as *select r.a from r where r.a not in (select * from s)*). It seems bizarre, to say the least, to use $\preceq_{\text{OWA}}$ while dealing with the CWA semantics. However, the alternative is to return no answers at all!

This is of course yet another illustration that returning certain answers as objects is not always possible, and one needs to return certain answers as knowledge instead. What sort of knowledge will depend, of course, on the representation system. If we use the set of $\exists\mathsf{Pos}$ formulae, then the best way of describing certain answers will be $A(1) \vee A(2)$.

Next, consider the same query under OWA. Then $Q(\llbracket D \rrbracket_{\text{OWA}}) = \{A \mid 3 \notin A\}$. Again, this is not representable by a single object: taking the greatest lower of $Q(\llbracket D \rrbracket_{\text{OWA}})$ results again in $\emptyset$, missing some valuable information. If we use representation systems that allow negations of atomic formulae, then $\neg A(3)$ will represent the certain answer properly.

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 the process of finding certain answers is reduced to

query evaluation, at both object and knowledge level. Furthermore, the knowledge level is crucial for obtaining results at the object 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. It also tells us that using the traditional way of finding certain answers only makes sense under OWA, and with restricted representation systems.
- In general, certain answers as knowledge give us more information than certain answers as objects.

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), sometimes even using database evaluation techniques (Liu and Levesque 2003). While not directly applicable to relational databases, the connection is worth studying, especially for representing certain answers as knowledge.

Another direction has to do with the detailed study of the efficiency of computing certain answers, and, if intractability is encountered, finding either tractable restrictions, or approximation schemes, perhaps in the spirit of (Reiter 1986).

Yet another direction is to apply the framework to non-relational models, particularly semi-structured and XML, for which incompleteness has been studied extensively (Abiteboul, Segoufin, and Vianu 2006; Barceló et al. 2010; Calvanese, De Giacomo, and Lenzerini 1998; David, Libkin, and Murlak 2010), and beyond, to graph data, where only preliminary results have been established so far (Barceló, Libkin, and Reutter 2014).

Acknowledgments I am grateful to Pablo Barceló and anonymous referees for their comments. Work partly supported by EPSRC grant J015377.

References

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

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

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

Arenas, M.; Barceló, P.; and Reutter, J. 2009. Query languages for data exchange: beyond unions of conjunctive queries. In *International Conference on Database Theory (ICDT)*, 73–83.

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

Barceló, P.; Libkin, L.; Poggi, A.; and Sirangelo, C. 2010. XML with incomplete information. *Journal of the ACM* 58(1).

Barceló, P.; Libkin, L.; and Reutter, J. 2014. Querying regular graph patterns. *Journal of the ACM* 61(1).

Barwise, J., and Feferman, S., eds. 1985. *Model-Theoretic Logics*. Springer Verlag.

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

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

Chang, C., and Keisler, H. 1990. *Model Theory*. North Holland.

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

David, C.; Libkin, L.; and Murlak, F. 2010. Certain answers for XML queries. In *ACM Symposium on Principles of Database Systems (PODS)*, 191–202.

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

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

Gheerbrant, A.; Libkin, L.; and Sirangelo, C. 2013. When is naïve evaluation possible? In *ACM Symposium on Principles of Database Systems (PODS)*, 201–212.

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

Gunter, C. 1992. *Semantics of Programming Languages: Structures and Techniques*. MIT Press.

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

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

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

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

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

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

Liu, Y., and Levesque, H. J. 2003. A tractability result for reasoning with incomplete first-order knowledge bases. In *IJCAI*, 83–88.

Reiter, R. 1980. Equality and domain closure in first-order databases. *Journal of the ACM* 27(2):235–249.

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

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

Rossman, B. 2008. Homomorphism preservation theorems. *Journal of the ACM* 55(3).

Suciu, D.; Olteanu, D.; Re, C.; and Koch, C. 2011. *Probabilistic Databases*. Morgan&Claypool Publishers.