Certain Answers as Objects and Knowledge*

Leonid Libkin[a,*]

[a] *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 counterintuitive 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 values 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.

Keywords: incomplete information, database queries, certain answers, data models, certain knowledge, open and closed world, efficient computation

1. Introduction

Handling incomplete information is one of the oldest topics in data management research. It has been tackled both from the database perspective, resulting in classical notions of the semantics and complexity of query evaluation [1, 2], and from the AI perspective, providing an alternative view of the problem, see, e.g., [3, 4]. 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, particularly those where techniques from both the data management community and the knowledge representation community have been heavily used. These include data integration [5], data exchange [6], ontology-based data access [7, 8], inconsistent databases [9], probabilistic data [10], and data quality [11]. Here we treat the notion of "incompleteness" rather broadly: it means that data at our disposal does not provide a complete description, but rather suggests a number – perhaps infinite – of possible worlds. Most of the development here will need just this intuition,

*Preliminary version appeared in KR 2014.

*Corresponding author

Email address: libkin@inf.ed.ac.uk (Leonid Libkin)

but in concrete examples we shall use a common scenario of relational databases with missing (null) values [2, 3].

The central problem in all applications of incomplete databases is query answering. In the presence of incompleteness, one normally looks for *certain answers*: those that do not depend on the interpretation of unknown data. The concept was first mentioned in [12] and formally defined 35 years ago in [13] as follows. Assume that the semantics $[\![D]\!]$ of an incomplete database D is given as a set of complete databases, or possible worlds, D'. These are databases which the incomplete 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 data that was not completely specified.

If D is a relational, or an XML, or a graph database, $[\![D]\!]$ is usually obtained by replacing nulls (missing values) with real values, and perhaps adding extra information, such as tuples that were not in the database. In other applications, the definition of $[\![D]\!]$ varies, but the notion of certain answers does not. We now give a few examples.

Data integration. Here D is a source database (in fact it may contain multiple sources) and we need to answer queries over an integrated global schema database. The integration process is guided by a schema mapping M relating the schema of the source with that of the global schema over which queries must be answered. The process is virtual: the integrated database is not built, as M rarely makes it unique. Instead, D and M together provide an incomplete description of the integrated database, i.e.,

$$[\![D]\!]_M = \{D' \text{ of the global schema } \mid \; D \text{ and } D' \text{ satisfy } M\}\,.$$

If a query Q is posed against the global schema database, and we have access to the source D, query answers are typically defined as certain answers, i.e., $\bigcap\{Q(D') \mid D' \in [\![D]\!]_M\}$, see [14, 5].

Data exchange. This scenario is similar to the previous one: we have a schema mapping M between two schemas (usually called source and target schemas in this context), and still want to compute certain answers defined just as above. The difference is that this time the target schema database is materialized, and certain answers must be found based on a specific instance containing data exchanged between the source and the target, see [6, 15].

Consistent query answering. Assume we have a database D and a set Σ of integrity constraints over it, such as keys, foreign keys, etc, that D is supposed to satisfy. An inconsistent database fails to satisfy Σ, so one looks for its repairs D'; those are smallest changes that restore consistency (these can be defined in a variety of ways [9]). Then $[\![D]\!]_\Sigma$ consists of all such repairs, and for a given query Q one looks for consistent query answers defined as $\bigcap\{Q(D') \mid D' \in [\![D]\!]_\Sigma\}$. These are query answers that are true in all the repairs that restore consistency.

Ontology based query answering. An ontology Θ provides additional information about an incomplete database. Together, a database D and an ontology Θ define a set $[\![D]\!]_\Theta$ of possible worlds that are completions of the database D that make it satisfy all the ontology constraints. Ontology-based query answering – a very active research theme as of late – boils down to taking a query Q and finding certain answers to it over $[\![D]\!]_\Theta$, see [7, 8, 16].

Thus, in all these applications, spanning a large spectrum of data management and knowledge representation tasks, certain answers are the standard way of defining answers to queries. While the exact semantics of possible worlds changes, the definition of certain answers stays intact.

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 leads to many problems, and crucially to producing meaningless query answers.

To argue that this is the case, and to explain basic ideas behind the approach we present, 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 $Q([\![D]\!])$. The definition of $\mathsf{cert}_\cap$ uses the intersection operator as a means of extracting such common information.

Now take a simple example: we have a relation $R = \{(1,2),(3,\bot)\}$ in a database D, 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 [3], 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 domain of 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 [2] and as logical theories [3], rather than treat them separately. In addition, we bring in ideas from another approach, based on *ordering* incomplete

databases in terms of their informativeness [17, 18]. Combining these three approaches, taking, essentially, only what is really needed from each one of them, allows us to define certain answers in a new way that avoids some of the problems exhibited by the standard accepted definition.

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".

We concentrate here on the general problem of answering queries over incomplete databases, rather than specific applications, such as those listed earlier. In fact, in several of those applications the query answering problem is reduced to running a query on a specific incomplete database. Our main technical contributions are as follows.

1. Query answering is then 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, and yet more disciplined than what is called weak representation systems (which define certain information exclusively by means of intersection). Representation systems let one define important sets of objects by logical formulae, in the spirit of [3].

2. Our key result is that under the right choice of the 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. Even when such an evaluation is expensive, the approach suggests how to look for approximating query answers.

   The right choice of the semantics relies on the notion of informativeness and simply says that one cannot get less informative answers on more informative inputs (indeed, if we have an input I to a query Q, and then we gain extra information about I and obtain a more informative instance I', at the very least we could just go back to I and evaluate Q on it; hence, gaining information about the input can only make query output more informative). This very basic principle however has been ignored by most of the research on incompleteness in databases; we shall explain later why this happened.

3. We also show that 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. Also, even when both exist, we need certain answers as knowledge to prove that certain answers as objects are correct. Thus, representing certain knowledge about query answers is not a mere convenience, it may well be a necessity – even if not seen by the end user, it is necessary to provide correctness guarantees.

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 4. Section 5 defines representation systems and connects knowledge certainty with greatest lower bounds. Section 6 defines certain answers for queries, and Section 7 shows how to compute them. Summary and future work are in Section 8.

2. Background: relational incompleteness

Incomplete databases. These have two types of values: *constants* (e.g., $1, 2, \ldots$) and *nulls*, representing unknown values. 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 [19, 2]. 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:

$$[\![D]\!]_{\text{OWA}} \;=\; \{v(D) \cup D' \mid v \text{ is a valuation and } D' \text{ is complete}\}\,,$$

where D' is over the same vocabulary as D, and union is taken relation-by-relation. We shall also consider the *weak* CWA, or WCWA semantics, inspired by [20], given by

$$[\![D]\!]_{\text{WCWA}} \;=\; \{v(D) \cup D' \mid v \text{ is a valuation and } \mathrm{adom}(D') \subseteq \mathrm{adom}(v(D))\}\,.$$

That is, under CWA, we simply instantiate nulls by constants; under OWA, we can also add arbitrary new tuples (the database D'), and under WCWA, we can only add new tuples formed by elements already present (which is expressed by the condition $\mathrm{adom}(D') \subseteq \mathrm{adom}(v(D))$). For instance, if $D_0 = \{(\bot, \bot')\}$, then $[\![D]\!]_{\text{CWA}}$ only has instances $\{(c, c')\}$ for $c, c' \in \mathsf{Const}$, 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.

If one has a query Q to be evaluated on an incomplete database D, one needs to represent the set

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

of possible answers to Q on complete databases that are represented by D. One possibility is to look for an answer A so that $[\![A]\!] = Q([\![D]\!])$, i.e., A denotes exactly the answers to Q over $[\![D]\!]$. When such an answer A exists for every query Q from a given class of queries, one talks about *strong representation systems* for that class, under the semantics $[\![\,]\!]$, see [19, 2]. Unfortunately, strong representation systems may not exist even for simple queries over standard semantics of incompleteness, e.g., even quantifier-free queries for the closed-world semantics [2].

Classical definition of certain answers. In an attempt to overcome the problem of non-existence of strong representation systems, [2] proposed a notion of *weak* representation systems, whose idea, in essence, is that only the *certain* information in $Q([\![D]\!])$ and A should be the same. In the process of defining it formally, they introduced the notion of certain answers, that has since dominated the literature of query answering over incomplete databases.

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

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

Note that $\mathsf{cert}_\cap(Q, D)$ 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 $\mathsf{cert}_\cap(Q, D)$ 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 [1].

Logics. Most database query languages are based on *first-order predicate logic*, or FO. Its formulae, over a relational vocabulary, are built from relational atoms $R(\bar{x})$, where R is a vocabulary symbol, and equational atoms $x = y$, by closing them under the Boolean connectives $\wedge, \vee, \neg$ and quantifiers $\exists$ and $\forall$. Free variables of a formula φ are defined in the standard way. If $\bar{x}$ is the tuple of free variables of φ, we may indicate this explicitly by writing $\varphi(\bar{x})$. For FO formulae in this paper we use the standard semantics, even when structures mix elements from Const and Null. Recall that these sets are disjoint, and hence every equality between a constant and a null will evaluate to false. Equalities between nulls evaluate to true if nulls are the same. Using this standard FO semantics (i.e., not looking at the semantic distinction between constants and nulls) is common in databases and is often referred to as naïve evaluation. That is, a query is evaluated by the standard query evaluation engine, despite the fact that some entries are unknown.

We shall deal with several fragments of FO, of importance both in logic and in databases.

- *Conjunctive queries* are the $\exists, \wedge$-fragment of FO, i.e., it has no disjunction, negation, and universal quantification. Such formulae are of the form $\varphi(\bar{x}) \;=\; \exists \bar{y}\, R_1(\bar{u}_1) \wedge \ldots \wedge R_k(\bar{u}_k)$, where the R_is are relation symbols from the vocabulary, and each tuple $\bar{u}_i$, whose length equals the arity of R_i, contains variables from $\bar{x}, \bar{y}$. One can also add explicit equalities between variables in conjunctive queries (they do not add expressiveness of course).

- The class of *existential positive formulae*, denoted by $\exists$Pos, is the $\wedge, \vee, \exists$-fragment of FO (i.e., it disallows negation and universal quantification). In terms of their expressiveness, $\exists$Pos formulae correspond precisely to *unions of conjunctive queries*, i.e., disjunctions of conjunctive queries, although $\exists$Pos formulae can be more succinct.

- The class of *positive formulae*, denoted by Pos, is the $\wedge, \vee, \exists, \forall$-fragment of FO (i.e., it disallows negation).

- Formulae *true* and *false* belong to all these classes.

It is known that $\mathsf{cert}_\cap(Q, D) = Q^{\mathsf{C}}(D)$ for $\exists$Pos queries under OWA [2], and for Pos queries under WCWA [21]. There is a fragment for which the equality holds under CWA as well, and it will be given later in Section 5.

3. Problematic certain answers

The standard treatment of incompleteness in commercial DBMSs (in particular SQL's treatment of nulls) has been heavily criticized in the past [22, 23, 24], but the theoretical approaches have so far been spared. The goal of this section, which serves as additional motivation for our re-thinking of the standard approaches to certain answers, is to re-examine some of the most basic theoretical notions related to handling incompleteness in databases, and to demonstrate their severe shortcomings.

Semantics of query answers. Let us look at the seemingly uncontroversial definition saying that if we are lucky enough to get A satisfying $[\![A]\!] = Q([\![D]\!])$, then A should be viewed as the answer to Q on D. At the first glance it looks like a reasonable condition, but nonetheless there is one questionable assumption built into it. Note that this definition requires that both the input database D, and the answer A, be interpreted under the *same* semantics $[\![\,]\!]$. However, a priori, there is no reason for it. Why, for instance, should the answer to a query be interpreted under CWA if this is the semantics of the input? And what if we have data of different formats, e.g., XML-to-relational or relational-to-XML queries?

The importance of intersection, and certainty of certain answers. The idea of using certain answers given as intersections of all query answers initially came from weak representation systems of [2]. If $[\![A]\!] = Q([\![D]\!])$ is too strong a condition, one tries to replace it by $[\![A]\!] \sim Q([\![D]\!])$, where $\sim$ is some equivalence relation. In the definition of [2], the relation is defined as follows. Given two sets of database instances, $\mathcal{I}_1$ and $\mathcal{I}_2$, and a query language $\mathcal{L}$, we let $\mathcal{I}_1 \sim_{\mathcal{L}} \mathcal{I}_2$ if $\bigcap\{q(D') \mid D' \in \mathcal{I}_1\} = \bigcap\{q(D') \mid D' \in \mathcal{I}_2\}$ for each query q in $\mathcal{L}$.

This equivalence relation looks quite ad hoc. It was defined that way to ensure compositionality of queries, and what really survived from it in applications is the idea of intersection for certain answers. But that is problematic for two reasons. First, there are models other than relational. What can one do, for instance, for XML queries returning documents? (A side remark: much of the work on incompleteness in XML has been restricted to XML-to-relational queries, for this very reason [25, 26, 27].) But even more importantly, we may lose information by taking intersection and removing tuples from the answer. Indeed, taking tuples away amounts to removing *data*, not information. In fact, the process can actually *add* information: for instance, under CWA, by removing a tuple we gain information that it is not in the answer, as discussed in the introduction. Hence, certain answers defined by the intersection operator cannot be called certain in all scenarios.

Representation of query answers. We are used to queries returning database objects – tables, XML documents, graphs. But are these sufficiently expressive to describe answers on incomplete databases? Specifically, are these sufficiently expressive to represent sets $Q([\![D]\!])$? Such sets may well be infinite, and describing them may require a more complex representation mechanism than simple database objects. Some attempts have been made, for instance with conditional tables [2] which "look" like relations and give a strong representation system. There is a price to pay: they correspond to a rather bizarre fragment of FO, and have intractable complexity even for simple queries [1]. But is it really necessary to have representations of answers that look like database relations, while they are not?

The approach we now present will try to address these lines of criticism, and will provide a new and more disciplined version of certain answers.

4. Objects, knowledge, and information ordering

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 minimalistic setting inspired by abstract model theory [28] or information systems [29], with some specific features tailored to handle incompleteness, as also used in [30, 21]. A word on notation: in our abstract setting, we shall refer to objects by letters x, y, etc., while dealing with relational databases (which serve as the main example here) we shall use D, D', etc.

The key elements of this minimalistic settings are:

- the notion of a *database pre-domain*, capturing the basic intuition that database objects can be complete or incomplete, and the semantics of an incomplete object is a set of possible complete worlds for it;
- the notion of an *information ordering* capturing the intuition that some objects are more informative than others; and
- the notion of a *pre-representation system* that introduces formulae that can be satisfied by objects.

We now formalize these as follows.

Definition 1 (Pre-domain). *A* database pre-domain *is a triple* $\mathbb{D}^\circ = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!] \rangle$*, where* $\mathcal{D}$ *is a set,* $\mathcal{C} \subseteq \mathcal{D}$*, and* $[\![\,]\!]$ *is a function from* $\mathcal{D}$ *to* $2^{\mathcal{C}}$*, such that* $c \in [\![c]\!]$ *for every* $c \in \mathcal{C}$ *and* $[\![c]\!] \subseteq [\![x]\!]$ *whenever* $c \in [\![x]\!]$*.*

The intuition is as follows:

- $\mathcal{D}$ 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 the semantic function: $[\![x]\!] \subseteq \mathcal{C}$ is the semantics of an incomplete database x, i.e., the set of all complete databases that x can represent.

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.

Definition 2 (Information ordering). *Given a semantics* $[\![\,]\!]$ *of incompleteness, an* information ordering *associated with it is given by*

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

The intuition is that y is at least as informative as x. Indeed, the more possible objects we have in the semantics of an incomplete object, the less we know about it. In the extreme case, when we know nothing about an object, its semantics is everything, i.e., it can denote any other object. Note that $\preceq$ is a preorder, i.e., it is reflexive and transitive, but both $x \preceq y$ and $y \preceq x$ may be true for equivalent objects x and y, i.e., if $[\![x]\!] = [\![y]\!]$.

Note that the last condition in the definition of pre-domain simply says that if $c \in [\![x]\!]$, then $x \preceq c$, i.e., an incomplete object is less informative than the objects it represents.

Definition 3 (Pre-representation system). *A* pre-representation system *is a triple* $\mathbb{RS}^\circ = \langle \mathbb{D}^\circ, \mathbb{F}, \models \rangle$*, where* $\mathbb{D}^\circ = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!] \rangle$ *is a predomain,* $\mathbb{F}$ *is a set, and* $\models$ *is a relation between* $\mathcal{D}$ *and* $\mathbb{F}$ *(written as* $x \models \varphi$ *for* $x \in \mathcal{D}$ *and* $\varphi \in \mathbb{F}$*) such that* $x \preceq y$ *and* $x \models \varphi$ *imply* $y \models \varphi$*.*

The intuition is that $\mathbb{F}$ is a set of *formulae* that express knowledge we possess about objects in $\mathcal{D}$, and $\models$ is the satisfaction relation between objects in $\mathcal{D}$ and formulae in $\mathbb{F}$. The last condition says that 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, that is, $\{\varphi \mid x \models \varphi\}$ and $\mathsf{Mod}(\varphi)$ for models of φ, that is, $\{x \mid x \models \varphi\}$. These are extended to sets in the usual way:

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

It is well known that Mod and Th define a Galois connection between $\mathcal{D}$ and $\mathbb{F}$. That is, both Mod and Th are anti-monotone, $X \subseteq \mathsf{Mod}(\mathsf{Th}(X))$ and $\Phi \subseteq \mathsf{Th}(\mathsf{Mod}(\Phi))$ for all X and Φ. In particular, this implies that $\mathsf{Mod}(\mathsf{Th}(\cdot))$ is a closure operator.

4.1. 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. We now present two ways of extracting certain information from $X \subseteq \mathcal{D}$.

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 exists, we denote it by $\Box_{\mathcal{O}} X$. The subscript $\mathcal{O}$ is for 'object'.

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 $\Box_{\mathcal{K}} X$. The subscript $\mathcal{K}$ is for 'knowledge'.

Note that $\Box_{\mathcal{K}} X$ is a formula in $\mathsf{Th}(X)$: indeed, since $\mathsf{Mod}(\mathsf{Th}(\cdot))$ is a closure operator, $X \subseteq \mathsf{Mod}(\mathsf{Th}(X)) = \mathsf{Mod}(\Box_{\mathcal{K}} X)$.

Thus, certain information contained in X is described

- 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 of 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 much simplified version of the framework was used in [30] and in [31] for XML documents. Crucially, the framework lacked the notion of representation systems, as there were no formulae defined on database domains. A rudimentary case of $\Box_{\mathcal{K}} X$ was used there, by identifying what we view as the set $\mathbb{F}$ of formulae here with $\mathcal{D}$ itself. If $x \models y$ means $y \preceq x$, then analog of Theorem 1 below for that simple case is a consequence of results in [30]. An attempt to give conditions for query evaluation to compute certain answers was made there as well, but without representation systems it required some rather ad hoc conditions on complete objects and was not satisfactory.

Example 1. 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 violate the definition of $\Box_{\mathcal{K}} X$ in this case. We shall continue with this example later in Example 2 and use the machinery developed in the paper to formally prove the results stated here. We shall also present a formula $\Box_{\mathcal{K}} X$ under CWA that comes from a more expressive subclass of FO.

5. Representation systems

So far we imposed no conditions on pre-domains and 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. In real life, we have (potentially) infinite domains of database elements. If we have a database with the tuple $(1, 2)$, we should expect to have a database with the tuple $(2, 3)$, or $(3, 4)$, and so on. The condition that we introduce now captures this intuition in our abstract model.

To see how this can be done, consider an arbitrary relational database D, and a renaming f of elements of its active domain, i.e., a one-to-one mapping $f : \text{adom}(D) \to U$ for some set U. If $f(D)$ is the database obtained by replacing each element $a \in \text{adom}(D)$ with $f(a)$, then databases D and $f(D)$ are *isomorphic*: a tuple $\bar{t}$ is in a relation R of D iff $f(\bar{t})$ is in the same relation R in D'. Note that such a map does not distinguish nulls and constants. Two isomorphic databases will agree on logical formulae not mentioning constants. Even if we have a logical formula mentioning constants from a finite set C, databases D and $f(D)$ will agree on it as long as $f(a) = a$ for every $a \in C$.

The intuition behind the 'enough objects' condition is that for every such renaming f of elements of the active domain, $f(D)$ is a legitimate database, and D and $f(D)$ agree on formulae that only mention constants preserved by f.

To formulate this at the abstract level of database pre-domains, consider the relation $D \approx_C D'$ saying that there is a one-to-one mapping f preserving C such that $D' = f(D)$. It is easily seen to be an equivalence relation; moreover, if the range of f only includes constants, then $f(D)$ is a complete database that is in $[\![D]\!]$ for the standard semantics like OWA, CWA, WCWA. Finally, if $D \approx_{C \cup C'} D'$, then $D \approx_C D'$ and $D \approx_{C'} D'$. These conditions are easy to state at the level of database pre-domains.

Definition 4 (Database domain). *A* database domain $\mathbb{D}$ *as a tuple* $\langle \mathcal{D}, \mathcal{C}, [\![\,]\!], Iso \rangle$ *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 both* $x \approx_j y$ *and* $x \approx_{j'} y$*.*

Coming back to the relational intuition, the relations $\approx_j$ are exactly the relations $\approx_C$, where C ranges over finite sets of constants. The first condition says that we can replace nulls by constants outside C (since $\mathsf{Const} - C$ is infinite); the second one was explained earlier.

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 $\mathsf{Mod}(\varphi \wedge \psi) = \mathsf{Mod}(\varphi) \cap \mathsf{Mod}(\psi)$. This is just a technical condition so that we could represent a finite set $\{\varphi_1, \ldots, \varphi_n\}$ of formulae by a single formula, namely their conjunction $\varphi_1 \wedge \ldots \wedge \varphi_n$.

The second assumption is that the formulae have something to do with the intended meaning of incomplete information. For this we require that some basic sets of objects be definable by formulae. In our minimalistic model the most basic sets are of the form $[\![x]\!]$. Thus, we shall require their definability; since $[\![x]\!] \subseteq \mathcal{C}$, we shall require the existence, for each object x, of a formula δ_x such that $\mathsf{Mod}(\delta_x) \cap \mathcal{C} = [\![x]\!]$. We shall see that this is the same as asking that $\mathsf{Mod}(\delta_x) = \uparrow x$, where $\uparrow x = \{y \mid x \preceq y\}$ is the set of objects at least as informative as x. Note that we simply require their existence, although we shall see that for basic relational semantics of incompleteness they will be easy to construct explicitly.

All these are summarized in the following definition.

Definition 5 (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* $\delta_x \in \mathbb{F}$ *with* $\mathsf{Mod}(\delta_x) \cap \mathcal{C} = [\![x]\!]$*;*
- *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.

5.1. 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 were already explained. They are indexed by finite subsets of Const, and for such a finite set C, we have $D \approx_C D'$ iff there is an isomorphism f between D and D' such that both f

and f^{-1} are identity on C. More precisely, f is a one-to-one mapping from $\mathrm{adom}(D)$ to $\mathrm{adom}(D')$ so that $f(a) = f^{-1}(a) = a$ if $a \in C$, and $\bar{t} \in R^D$ iff $f(\bar{t}) \in R^{D'}$.

We write $\preceq_*$ for the information ordering given by $[\![\,]\!]_*$, i.e., $D \preceq_* D'$ iff $[\![D']\!]_* \subseteq [\![D]\!]_*$. These orderings $\preceq_{\mathrm{OWA}}, \preceq_{\mathrm{CWA}}$, and $\preceq_{\mathrm{WCWA}}$ are known to be expressible in terms of homomorphisms. Given two relational databases D and D', a homomorphism $h : D \to D'$ is a map from $\mathrm{adom}(D)$ to $\mathrm{adom}(D')$ such that $h(c) = c$ for each constant c, and such that for every relation symbol R and 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 $\mathrm{adom}(h(D)) = \mathrm{adom}(D')$, and *strong onto* if $D' = h(D)$.

It was shown in [30, 21] that $D \preceq_* D'$ iff there exists a

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

In what follows, we describe sets of formulae $\mathbb{F}$ and formulae δ_D for each D. Let $\mathrm{PosDiag}(D)$ 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 $\mathrm{PosDiag}(D) = R(1,2) \wedge R(2,x_1) \wedge R(x_1,x_2)$.

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

WCWA. The WCWA representation system is $\mathbb{RS}_{\mathrm{WCWA}}(\sigma) = \langle \mathbb{D}_{\mathrm{WCWA}}(\sigma), \mathsf{Pos}, \models\rangle$. 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

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

CWA. We need to define an extension of the class of positive formulae, introduced by [32] and used recently in [21]. 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 logical constants *true* and *false* and 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 $\alpha(\bar{x})$ is an atomic formula, then
$$\forall \bar{x}\ (\alpha(\bar{x}) \to \varphi(\bar{x}, \bar{y}))$$
is a $\mathsf{Pos}^{\forall\mathsf{G}}$ formula.

That is, such formulae are either $\forall x_1, x_2\ (x_1 = x_2 \to \varphi(x_1, x_2, \bar{y}))$ for equational atoms, or $\forall \bar{x}\ (R(\bar{x}) \to \varphi(\bar{x}, \bar{y}))$, where R is of arity $|\bar{x}|$.

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

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

where $\bar{y} = (y_1, \ldots, y_m)$ is a tuple of variables whose length m is the arity of R, and for a tuple $\bar{t} = (a_1, \ldots, a_m)$ in R^D we write $\bar{y} = \bar{t}$ for $(y_1 = a_1) \wedge \ldots \wedge (y_m = a_m)$.

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. 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, and the fact that $\exists\mathsf{Pos}$, Pos and $\mathsf{Pos}^{\forall\mathsf{G}}$ formulae are preserved under homomorphisms (respectively onto and strong onto homomorphism), see [33, 34] for $\exists\mathsf{Pos}$ and Pos and [32, 21] for $\mathsf{Pos}^{\forall\mathsf{G}}$. And the properties of δ_D are again straightforward from the definition of the semantics, as they simply state it in the language of FO. □

5.2. Properties of representation systems

We now collect some basic properties of representation systems, essentially the toolkit that will be useful later. We look at certain information contained in sets $[\![x]\!]$ and prove that, as expected, it is represented at the object level by x itself, and at the knowledge level by δ_x, defining the semantics of 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 the 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 the sentence $\exists x\ (x = 1)$) and hence there are potentially formulae in $\mathsf{Th}([\![x]\!]_{\approx_j})$ which are not satisfied by x. However, the following holds.

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, for every object* x*:*

1. $\mathsf{Mod}(\delta_x) = \uparrow x$;
2. $\delta_x = \Box_{\mathcal{K}}[\![x]\!]$.

Proof. To show the first item, choose $j \in J$ so that two objects related by $\approx_j$ agree on δ_x. Since $[\![x]\!]_{\approx_j}$ is not empty, pick an object c from this set. Then $c \models \delta_x$ since $c \in [\![x]\!]$. Since c and x agree on δ_x, we have $x \models \delta_x$. By the monotonicity of formulae we conclude $\uparrow x \subseteq \mathsf{Mod}(\delta_x)$. Suppose we have an object y such that $x \not\preceq y$ and $y \models \delta_x$. Then $[\![y]\!] \not\subseteq [\![x]\!]$ and thus we have a complete object $c \in [\![y]\!] - [\![x]\!]$. Since $y \preceq c$, we have $c \models \delta_x$, contradicting $\mathsf{Mod}(\delta_x) \cap \mathcal{C} = [\![x]\!]$. This shows $\mathsf{Mod}(\delta_x) \subseteq \uparrow x$ and thus $\mathsf{Mod}(\delta_x) = \uparrow x$.

To prove the second item, we start by showing 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 remark that the condition $\mathsf{Mod}(\delta_x) = \uparrow x$ could alternatively be used to define formulae δ_x, since $\mathsf{Mod}(\varphi) = \uparrow x$ implies $\mathsf{Mod}(\varphi) \cap \mathcal{C} = [\![x]\!]$. Indeed, consider any complete object c that satisfies φ. Then $x \preceq c$ and we have $c \in [\![c]\!] \subseteq [\![x]\!]$ by the properties of representation systems.

Corollary 1 says that for sets X of the form $[\![x]\!]$ we have $\Box_{\mathcal{O}}X \models \Box_{\mathcal{K}}X$. We shall see later that the same is true for the important sets of the form $Q([\![x]\!])$ that we need for defining certain answers. But in general, this condition need not hold. 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 knowledge may be preferable, as it conveys more information.

Proposition 5. *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 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 6. *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. Assume the conditions in 1. hold and let $\Phi_0 = \{\varphi_1, \ldots, \varphi_m\}$ axiomatize $\mathsf{Th}(x)$. Take δ_x to be $\varphi_1 \wedge \ldots \wedge \varphi_m$. Then $\mathsf{Mod}(\delta_x) = \mathsf{Mod}(\mathsf{Th}(x))$. Suppose $y \succeq x$. If $\varphi \in \mathsf{Th}(x)$ then $y \models \varphi$ and thus $y \in \mathsf{Mod}(\mathsf{Th}(x))$. Conversely, let $y \not\succeq x$. Then we have $\psi \in \mathsf{Th}(x) - \mathsf{Th}(y)$ and thus $y \notin \mathsf{Mod}(\mathsf{Th}(x))$. Hence $\uparrow x = \mathsf{Mod}(\mathsf{Th}(x)) = \mathsf{Mod}(\delta_x)$.

If we have δ_x for each x, then $\mathsf{Mod}(\mathsf{Th}(x)) = \uparrow x$. Indeed, assume $x \preceq y$; then $y \models \mathsf{Th}(x)$ by monotonicity. If $x \not\preceq y$, then $y \not\models \delta_x$ while $x \models \delta_x$, so $y \not\models \mathsf{Th}(x)$. By the definition of δ_x this implies $\mathsf{Mod}(\mathsf{Th}(x)) = \mathsf{Mod}(\delta_x)$, with δ_x axiomatizing $\mathsf{Th}(x)$. Also if $\mathsf{Th}(x) \subseteq \mathsf{Th}(y)$, then $\delta_x \in \mathsf{Th}(y)$, hence $y \models \delta_x$ and thus $x \preceq y$. □

5.3. Certain knowledge 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 entailment: $\Phi \vdash \Psi$ 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 $\alpha = \bigwedge \mathsf{Th}(X)$. □

6. Defining certain answers to queries

Now we move to answering queries. We take an abstract view of queries, introduced in [35] and standard in database literature [19], when a query is viewed a mapping Q that takes an object and returns another object. In real life queries are written in a specific query language, but their semantics is of course such. For instance, relational queries take relational databases and return relational databases (most commonly, single relations), while they can be written in many languages such as FO, SQL, datalog, etc.

Thus, for two database domains $\mathbb{D} = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!], \text{Iso} \rangle$ and $\mathbb{D}' = \langle \mathcal{D}', \mathcal{C}', [\![\,]\!]', \text{Iso}' \rangle$, 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 that they *preserve informativeness*, which can be stated as follows:

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 of inputs and outputs (technically, they mean 'knowing at least as much'). 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. Thus, blindly using some fixed semantics for query results – as in fact is often done – does not necessarily make sense.

Formally, this notion is defined as follows.

Definition 6 (Preserving informativeness). *A query* $Q : \mathbb{D} \to \mathbb{D}'$ preserves informativeness *if it is monotone with respect to the information orderings given by the semantics of query inputs and query answers, i.e.,*

$$x \preceq y \quad \textit{implies} \quad Q(x) \preceq' Q(y).$$

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 latter, 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 notions of certain answers:

- 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 [2]. So the next attempt is to find a formula $\varphi_{Q,D}$ in some logical formalism so that

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

When this happens, one refers to such a logical formalism as a *strong representation system* (see [19, 2]), 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 [2]. Syntactically, this is quite heavy, does not correspond to any natural fragment of FO, 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: 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 the 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 $\langle \mathbb{D}(\sigma), \exists\mathsf{Pos}, \models \rangle$. Since $\exists\mathsf{Pos}$ formulae are monotone, there is no $\varphi \in \exists\mathsf{Pos}$ with $\mathsf{Mod}(\alpha) = Q([\![D]\!]_{\text{CWA}})$. But there an $\exists\mathsf{Pos}$ formula φ one such that $\mathsf{Mod}(\varphi) = \mathsf{Mod}(\mathsf{Th}(Q([\![D]\!]_{\text{CWA}})))$. In fact, the obvious answer $\varphi = A(1) \vee A(2)$ does the job (we use predicate $A(\cdot)$ for 'answer'). To see this, it suffices to note that $\exists\mathsf{Pos}$ formulae, which are disjunctions of conjunctive queries, are in $\mathsf{Th}(Q([\![D]\!]_{\text{CWA}}))$ if they are disjunctions of conjunctive queries that are true in $\{1\}$, $\{2\}$, and $\{1, 2\}$. Those are conjunctive queries implied by the positive diagrams of such instances, i.e., $A(1)$, $A(2)$, and $A(1) \wedge A(2)$. From this one easily derives that $\mathsf{Mod}(\mathsf{Th}(Q([\![D]\!]_{\text{CWA}})))$ consists of instances containing either 1 or 2.

7. Computing certain answers

We now look at computing certain answers for queries. 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. That is, certain answer as object, or $\mathsf{cert}_{\mathcal{O}}(Q, x)$, is just $Q(x)$, while certain answer as knowledge, $\mathsf{cert}_{\mathcal{K}}(Q, x)$, is $\delta_{Q(x)}$.

Crucially, the former condition is a corollary of the latter: without a representation system for query answers, it 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 under very restricted scenarios. 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 preserve informativeness: $x \preceq y$ implies $Q(x) \preceq' Q(y)$. We need an additional condition of *genericity*, which is standard in the database context (see, e.g., [19]). Essentially, this condition says that queries applied to isomorphic objects return isomorphic objects. For instance, for queries expressible in first-order logic that do not refer to constants, if two databases D and D' are isomorphic, then so are the answers $Q(D)$ and $Q(D')$. If queries use constants, this is true when isomorphisms of databases leave those constants intact. In our abstract setting such isomorphisms are captured via relations in Iso. Hence, the analog of the genericity condition in this setting is as follows.

Definition 7 (Genericity). *A query $Q : \mathbb{D} \to \mathbb{D}'$ is* generic *if for every j, there is k so that $x \approx_k y$ implies $Q(x) \approx'_j Q(y)$.*

We now state the main result about computing certain answers, and after proving it discuss what it means for finding query answers, and what are the conditions imposed on queries and databases.

Theorem 2. *Let $Q : \mathbb{D} \to \mathbb{D}'$ be a query that preserves informativeness and is generic. Assume that there is a representation system $\mathbb{RS} = \langle \mathbb{D}', \mathbb{F}, \models \rangle$ over the domain of query answers. 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 $y \preceq' Q(x)$. Therefore, $Q(x)$ is the greatest lower bound of $Q([\![x]\!])$, that is, $Q(x) = \mathsf{cert}_{\mathcal{O}}(Q, x)$. □

7.1. Discussion

Theorem 2 says that the new definition of certain answers based on greatest lower bounds provides correctness guarantees. The conditions required for this are genericity and monotonicity of the query, as well as having a representation system for query answers. We now discuss the importance of these requirements, and how much 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. For instance, as the input domain we consider $\mathbb{D} = \{x, c_1, c_2\}$ such that c_1, c_2 are complete objects, incomparable with each other, and $x \prec c_1, c_2$ (of course $x \prec y$ means that $x \preceq y$ holds but $y \preceq x$ does not). In $\mathbb{D}'$, we have three complete objects c', c'_1, c'_2 and an incomplete object x', with the ordering $x' \preceq' c'$ and $c' \preceq' c'_1, c'_2$. Now define the query Q by $Q(x) = x'$, $Q(c_1) = c'_1$ and $Q(c_2) = c'_2$. Then $Q(x) \prec' \mathsf{cert}_{\mathcal{O}}(Q, x) = c'$. The existence of representation systems prevents this behavior. Indeed, if we had formulae δ, we would have had $\delta_{c'} \in \mathsf{Th}(Q([\![x]\!])) - \mathsf{Th}(Q(x))$, which is made impossible by (4) in the presence of a representation system.

Thus, it is essential to go via certain answers as knowledge to get the object representation. On the positive side, for common semantics (e.g., OWA and CWA), representation systems can easily be constructed.

Genericity. In many formalisms, this is essentially a "free" condition: genericity applies to most of the logical formalisms used for querying databases. There are exceptions, however, but the machinery developed here lets us handle those exceptions.

One of the exceptions is given by languages capable of referring to infinitely many constants. Such languages occur, for instance, in data exchange [6], where it is sometimes necessary to distinguish constants from nulls [36, 37], especially with rather expressive mappings. But then we can use another condition in place of genericity.

The condition is 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.

Proposition 7. *Let $Q : \mathbb{D} \to \mathbb{D}'$ be a query that preserves informativeness, 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 inclusion $\mathsf{Th}(Q(x)) \subseteq \mathsf{Th}(Q([\![x]\!]))$ 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 converse inclusion. □

Yet another situation where genericity is lost is when nulls and constants behave differently in queries. This is the standard behavior of SQL, for instance: any comparison involving a null results in a truth value *unknown* [24]. That is, checking $\bot_1 = \bot_2$ results in *unknown*, but of course under the mapping $\bot_1 \mapsto 1,\ \bot_2 \mapsto 2$ the comparison becomes *false*, which easily leads to non-genericity. In fact there are several variants of SQL-like evaluation procedures with nulls [38], and they all lack genericity.

However, in this case we get a simple corollary that provides us with some correctness guarantees.

Corollary 2. *If genericity is dropped as an assumption of Theorem 2, i.e., we have a query Q that preserves informativeness but is not necessarily generic, then $Q(x) \preceq' \mathsf{cert}_{\mathcal{O}}(Q, x)$, and $\delta_{Q(x)} \in \mathsf{Th}(Q([\![x]\!]))$.*

In other words, in this case the basic query evaluation $Q(x)$ provides us with an *approximation* of certain answers, since it gives us some, but perhaps not all, information that certain answers contain. Assuming that Q itself is efficient, this becomes an efficient approximation. In fact this idea was recently used to provide efficient evaluation of SQL queries with nulls that provides certainty guarantees [38].

Preserving informativeness. Monotonicity, or preserving informativeness, 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 outputs, which is often not justified. 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.

One possible approach to achieving monotonicity of queries is to insist on some fixed semantics of query answers, and then look at restrictions of queries under which monotonicity is achieved. This is essentially what approaches based on *naïve evaluation* [2, 21, 30] do. Indeed, Theorem 2 says that certain answers are computed by the evaluation of the query itself, and the goal of naïve evaluation is the same.

A different approach is to fix a language $\mathcal{L}$, and try to find a semantics of query answering so that all the queries in $\mathcal{L}$ are monotone. This has been explored to a lesser extent, although this is the direction that needs to be developed if we want to learn how to answer complex queries with certainty (since naïve evaluation inevitably imposes restrictions on queries).

We now offer one simple observation in this direction, showing that a defining certain answers as greatest lower bounds makes them behave reasonably. Namely, assume that a query Q is only defined on complete objects, and we want to extend it to all objects. The natural way to do this is to use certain answers as objects, i.e., the query $\mathsf{cert}_{\mathcal{O}}(Q)$ that sends each object x to $\mathsf{cert}_{\mathcal{O}}(Q, x)$. The following is immediate from the definitions.

Proposition 8. *For an arbitrary query Q defined on complete objects, the query $\mathsf{cert}_{\mathcal{O}}(Q)$ preserves informativeness.*

Proof. Let $x \preceq y$. Then $[\![y]\!] \subseteq [\![x]\!]$ and $Q([\![y]\!]) \subseteq Q([\![x]\!])$. Thus if greatest lower bounds of these sets are defined, we have $\bigwedge Q([\![x]\!]) \preceq' Q([\![y]\!])$, i.e., $\mathsf{cert}_{\mathcal{O}}(Q, x) \preceq' \mathsf{cert}_{\mathcal{O}}(Q, y)$. □

This proposition only confirms that certain answers, as defined here, behave logically. It does not give a recipe to compute them, but suggests new approaches which we shall discuss in the concluding section.

7.2. Certain answers for relational queries

We now look at examples of concrete domains and representation systems for relational databases. The domains of input databases 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?

To see what semantics was used, look at the classical certain answers, as defined in [2, 19], for Boolean queries. For an input database D, we have $\mathsf{cert}_\cap(Q, D) = \mathit{true}$ iff $Q(D') = \mathit{true}$ for every $D' \in [\![D]\!]$, and $\mathsf{cert}_\cap(Q, D) = \mathit{false}$ iff just one $Q(D')$ evaluates to *false*. Viewing this way of computing certain answers as the greatest lower bound corresponds to the ordering $\mathit{false} \preceq \mathit{true}$. Or, if *false* is represented by the empty set and *true* by the set $\{()\}$ containing the empty tuple, this corresponds to the subset ordering. Going to non-Boolean queries, we note that $\mathsf{cert}_\cap(Q, D)$ is the greatest lower bound in the same ordering $\subseteq$ on query answers that happen not to contain nulls.

In particular, this means that in the standard relational query answering over incomplete databases one assumes the *open world* semantics for query answers since tuples can be added. And note that the ordering $\preceq_{\text{OWA}}$ becomes $\subseteq$ when restricted to databases without nulls.

We thus look at relational queries as mappings $Q : \mathbb{D}_*(\sigma) \to \mathbb{D}_{\text{OWA}}(\sigma_m)$. The following proposition states that monotonicity (i.e., preservation of informativeness) for these classes is achieved for classes of FO formulae seen earlier.

Proposition 9. *Consider an m-ary relational query $Q : \mathbb{D}_*(\sigma) \to \mathbb{D}_{\text{OWA}}(\sigma_m)$. Then Q is preserves informativeness*

- *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.*

Proof. We need to show the following. Suppose $h : D \to D'$ is a homomorphism, and $\bar{a}$ is an m-tuple of elements of $\text{adom}(D)$ such that $\bar{a} \in Q(D)$. Then $h(\bar{a}) \in Q(D')$ if:

- Q is definable in $\exists\mathsf{Pos}$, or
- Q is definable in Pos and h is onto, or
- Q is definable in $\mathsf{Pos}^{\forall\mathsf{G}}$, and h is strong onto.

This is exactly the restatement of the required monotonicity. This in turn follows from known results saying that formulae in these classes are preserved under homomorphisms [33, 34, 32, 21]. Note that in our case homomorphisms are restricted as they preserve elements on Const but since we are interested in the syntax-to-preservation direction only, this restriction does not affect the proofs in [33, 34, 32, 21]. In fact the proofs just go by straightforward induction on the structure of the formula. □

This proposition, combined with Theorem 2, gives us the following.

Corollary 3. *Let Q be an m-ary relational query. If its answers are interpreted under the* OWA *semantics, then $\mathsf{cert}_{\mathcal{O}}(Q, D) = Q(D)$ provided that*

- *input databases are interpreted under* OWA *and Q is definable in $\exists\mathsf{Pos}$, or*
- *input databases are interpreted under* WCWA *and Q is definable in Pos, or*
- *input databases are interpreted under* CWA *and Q is definable in $\mathsf{Pos}^{\forall\mathsf{G}}$.*

For the "classical" way of defining certain answers, which involves throwing away tuples with nulls and computing intersection, the notion of *naïve evaluation* involves computing $Q(D)$ and throwing away tuples with nulls [2, 21]. Note that using the new notion of certain answers, namely $\mathsf{cert}_{\mathcal{O}}(Q, D)$, for the same classes of queries we no longer need to throw away tuples containing nulls: simply computing $Q(D)$ does the job. Returning to the example from the introduction, if we have a database D with a relation $R^D = \{(1, 2), (3, \bot)\}$ and a query Q that returns that relation, the classical naïve evaluation produces a single tuple $(1, 2)$, while $\mathsf{cert}_{\mathcal{O}}(Q, D)$ returns R^D itself, as expected.

The operator π^{C} that simply keeps all tuples that only use constants, is monotone (i.e., preserves informativeness) with respect $\preceq_{\text{OWA}}$. Thus, if we use the OWA semantics for answers, it is harmless, but still it may unnecessarily eliminate tuples from answers, as we have seen (not to mention that it costs us computationally). Furthermore, for other orderings, such as $\preceq_{\text{CWA}}$ and $\preceq_{\text{WCWA}}$, it is not even monotone, as the above example of $R^D = \{(1, 2), (3, \bot)\}$ shows. Thus, the use of the intersection operator is not necessary under OWA and is rather problematic under WCWA and CWA.

Example 2. We now show how Theorem 2, together with descriptions of representation systems for OWA and CWA, immediately yields results presented in Example 1.

Recall that we have a database D with a single relation R with tuples $(1, 2)$ and $(3, \bot)$. Consider the identity query Q (i.e., a query that simply returns R). We can view it as a query from $\mathbb{D}_{\text{OWA}}(\sigma_2)$ to $\mathbb{D}_{\text{OWA}}(\sigma_2)$; recall that σ_2 is the vocabulary of a single binary relation. For the representation system on $\mathbb{D}_{\text{OWA}}$ we use $\exists\mathsf{Pos}$ formulae, as before. Clearly Q is generic and preserves informativeness (i.e., is monotone), and thus $\Box_{\mathcal{K}}[\![D]\!]_{\text{OWA}} = \mathsf{cert}_{\mathcal{K}}(Q, D) = \delta_D$ by Theorem 2. The latter is the positive diagram of D as we saw, and hence $\Box_{\mathcal{K}}[\![D]\!]_{\text{OWA}} = \exists z R(1, 2) \wedge R(3, z)$, as was stated in Example 1.

Next we consider the query π^{C} eliminating tuples with nulls as a query from $\mathbb{D}_{\text{OWA}}(\sigma_2)$ to $\mathbb{D}^{\mathsf{C}}_{\text{OWA}}(\sigma_2)$, where the range of the latter is restricted only to databases without nulls. Again this query is monotone and generic, and it is easy to see that with the class of ground atoms and their conjunctions, $\mathbb{D}^{\mathsf{C}}_{\text{OWA}}(\sigma)$ forms a representation system. Hence Theorem 2 tells us that $\mathsf{cert}_{\mathcal{K}}(\pi^{\mathsf{C}}, D) = \Box_{\mathcal{K}}[\![D]\!]_{\text{OWA}} = \delta_{\pi^{\mathsf{C}}(D)}$ (with respect to the representation system based on conjunctions of ground atoms). Since the semantics remains to be OWA, we get $\delta_{\pi^{\mathsf{C}}(D)} = R(1, 2)$, again as stated in Example 1.

Finally, we can consider the identity query Q as a query from $\mathbb{D}_{\text{CWA}}(\sigma_2)$ to $\mathbb{D}_{\text{CWA}}(\sigma_2)$. We know what representation system to use for CWA: it is the one based on $\mathsf{Pos}^{\forall\mathsf{G}}$ formulae. Query Q is clearly monotone and generic, and hence again by Theorem 2 we have $\mathsf{cert}_{\mathcal{K}}(Q, D) = \Box_{\mathcal{K}}[\![D]\!]_{\text{CWA}} = \delta_D$, where the latter, in the $\mathsf{Pos}^{\forall\mathsf{G}}$ representation system is given by $\exists z \left(R(1, 2) \wedge R(3, z) \wedge \forall x, y\, R(x, y) \rightarrow \big((x = 1 \wedge y = 2) \vee (x = 3 \wedge y = z)\big)\right)$.

7.3. Closed world query answering

We have seen that under CWA, we have the largest class of queries for which certain answers can be found by straightforward query evaluation, applying Theorem 2. We now have a deeper look at $\mathsf{Pos}^{\forall\mathsf{G}}$ queries and provide more intuition about their power, both in terms of their logical expressiveness, and their procedural implementation in terms of relational algebra.

We first extend the class $\mathsf{Pos}^{\forall\mathsf{G}}$ syntactically. Define a class $\exists\mathsf{Pos}_0$ as follows:

- if x and y are distinct variables, then $x = y$ is in $\exists\mathsf{Pos}_0$;
- if $\bar{x}$ is an n-tuple of distinct variables and R is a relation symbol of arity n, then $R(\bar{x})$ is in $\exists\mathsf{Pos}_0$;
- if $\varphi_1(\bar{x})$ and $\varphi_2(\bar{x})$ are formulae in $\exists\mathsf{Pos}_0$, then $\psi(\bar{x}) = \varphi_1(\bar{x}) \vee \varphi_2(\bar{x})$ is in $\exists\mathsf{Pos}_0$;
- if $\varphi_1(\bar{x}_1)$ and $\varphi_2(\bar{x}_2)$ are formulae in $\exists\mathsf{Pos}_0$, and $\bar{x}_1, \bar{x}_2$ have no variables in common, then $\psi(\bar{x}_1, \bar{x}_2) = \varphi_1(\bar{x}_1) \wedge \varphi_2(\bar{x}_2)$ is in $\exists\mathsf{Pos}_0$;

- if $\varphi(\bar{x}, \bar{z})$ is a formula in $\exists\mathsf{Pos}_0$, then $\psi(\bar{x}) \;=\; \exists\bar{z}\, \varphi(\bar{x}, \bar{z})$ is a formula in $\exists\mathsf{Pos}_0$.

In other words, we restrict the class $\exists\mathsf{Pos}$ so that conjunction can only be taken when the conjuncts have no free variable in common, and disjunction when the free variables of disjuncts coincide.

Using this, we define an extension $\mathsf{Pos}^{\forall\mathsf{G}}_{ext}$ of the class $\mathsf{Pos}^{\forall\mathsf{G}}$ by changing the last formation rule from $\forall\bar{x}\ (\alpha(\bar{x}) \to \varphi(\bar{x}, \bar{y}))$ for atomic α by allowing the formulae $\alpha(\bar{x})$ to be from $\exists\mathsf{Pos}_0$.

Proposition 10. *The classes* $\mathsf{Pos}^{\forall\mathsf{G}}$ *and* $\mathsf{Pos}^{\forall\mathsf{G}}_{ext}$ *are equally expressive (i.e., for every formula of* $\mathsf{Pos}^{\forall\mathsf{G}}_{ext}$ *there is an equivalent formula of* $\mathsf{Pos}^{\forall\mathsf{G}}$ *and vice versa).*

Proof. To show this, one simply transforms the guarded formulae until the antecedents α become atomic formulae. This is done by using equivalences of

$$\forall\bar{x}_1, \bar{x}_2\ (\alpha_1(\bar{x}_1) \wedge \alpha_2(\bar{x}_2) \to \varphi(\bar{x}, \bar{x}_2, \bar{y})) \quad \text{and} \quad \forall\bar{x}_1\ \big(\alpha_1(\bar{x}_1) \to \forall\bar{x}_2\ (\alpha_2(\bar{x}_2) \to \varphi(\bar{x}_1, \bar{x}_2, \bar{y}))\big)$$

when $\bar{x}_1, \bar{x}_2$ share no variables, as well as of

$$\forall\bar{x}\ (\alpha_1(\bar{x}) \vee \alpha_2(\bar{x}) \to \varphi(\bar{x}, \bar{y})) \ \text{ and } \ \forall\bar{x}\ (\alpha_1(\bar{x}) \to \varphi(\bar{x}, \bar{y})) \ \wedge \ \forall\bar{x}\ (\alpha_2(\bar{x}) \to \varphi(\bar{x}, \bar{y}))$$

and

$$\forall\bar{x}\ (\exists\bar{z}\ \alpha(\bar{x}, \bar{z}) \ \to \ \varphi(\bar{x}, \bar{y})) \quad \text{and} \quad \forall\bar{x}, \bar{z}\ (\alpha(\bar{x}, \bar{z}) \to \varphi(\bar{x}, \bar{y})),$$

which are all readily verified. By applying these to a $\mathsf{Pos}^{\forall\mathsf{G}}_{ext}$ formula, we eventually put it into the $\mathsf{Pos}^{\forall\mathsf{G}}$ format, as follows from the definition of $\exists\mathsf{Pos}_0$. □

We can turn the above observation into a statement about the expressiveness of $\mathsf{Pos}^{\forall\mathsf{G}}$ queries at the *relational algebra* level, showing how much we can move away from the traditional unions of conjunctive queries under the CWA. Recall that relational algebra [19] consists of five operations: projection π, selection σ, cartesian product $\times$, union $\cup$, and difference $-$. The first four, known as *positive* relational algebra, capture exactly the power of $\exists\mathsf{Pos}$, or unions of conjunctive queries (hence also called select-project-join-union queries [19]). With the difference operator $-$, we get the full power of FO.

However, relational algebra often relies on some derived operations for implementing commonly occurring queries. When it comes to queries with *for-all* conditions, the most common such operation is *division* [19, 39]. If we have a relation R with attributes $A_1, \ldots, A_m, B_1, \ldots, B_k$ and a relation S with attributes $B_1, \ldots, B_k$, then $R \div S$ contains tuples of A-attributes of R that appear in R in every possible combination with a tuple from S, i.e., $R \div S = \{\bar{t} \in \pi_{\bar{A}}(R) \mid \forall\bar{s} \in S :\ (\bar{t}, \bar{s}) \in R\}$ (here $\pi_{\bar{A}}$ is the projection operator that only keeps attributes $A_1, \ldots, A_m$). It correspond to a very common class of queries, such as 'find students who take all classes'. The operation of division is of course expressible with $\sigma, \pi, \times, \cup, -$.

To show what the class $\mathsf{Pos}^{\forall\mathsf{G}}$ gives us in terms of relational algebra, we first define a class $\mathsf{RA}(\Delta, \pi, \times, \cup)$ of queries as follows. Let Δ be the query returning $\{(a, a) \mid a \in \mathrm{adom}(D)\}$; it is easily definable in positive relational algebra. Then $\mathsf{RA}(\Delta, \pi, \times, \cup)$ is the class of relational algebra queries obtained from base relations and Δ by closing them under π, $\times$, and $\cup$. Now we define RA_{CWA} as follows:

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

Now noticing that the guarded formulae used in the definition of $\mathsf{Pos}^{\forall\mathsf{G}}_{ext}$ correspond to the division operator, one can follow the definition of $\mathsf{Pos}^{\forall\mathsf{G}}_{ext}$ and obtain the following corollary of Proposition 10.

Corollary 4. $\mathsf{Pos}^{\forall \mathsf{G}}$ *and* $\mathsf{RA}_{\mathrm{CWA}}$ *are equally expressive.*

Consequently if input databases are interpreted under CWA and query outputs under OWA, as is very common, then $\mathsf{cert}_{\mathcal{O}}(Q, D) = Q(D)$ for all $\mathsf{RA}_{\mathrm{CWA}}$ queries. This gives so far the largest known class of relational algebra queries for which we can produce certain answers by simply evaluating queries on the input database, and it is expressed using well known operations of relational algebra, both basic and derived ones.

8. 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.

8.1. Future work

The next step is to use the general framework of this paper to provide efficient techniques for producing query answers with certainty guarantees. There are three directions one can pursue.

Efficient evaluation procedures for large classes of queries. So far, such techniques have mainly concentrated on finding classes of queries where naïve evaluation works [2, 21]. Our framework lets us shift focus and concentrate on finding proper query semantics to ensure the basic principle of preserving informativeness. One way to approach this is to use the fact that the certain answer query $\mathsf{cert}_{\mathcal{O}}(Q)$ preserves informativeness (Proposition 8). One can then look for evaluation procedures that are guaranteed to produce results which are below $\mathsf{cert}_{\mathcal{O}}(Q, D)$ in the informativeness ordering on the answers. If there is an efficient procedure with such properties, then it gives us an efficient way of producing query answers with certainty guarantees.

Similar ideas have been tried before, for relational algebra queries [40] and for SQL queries under the three-valued logic approach that SQL uses [38]. Approaches of [40, 38] were tailored for specific languages, and likewise had certainty guarantees tailored to specific requirements of those languages. Our general frameworks should let us find general schemes of generating efficient evaluation procedures for different languages and different correctness requirements.

Note also that the procedure of [38] for SQL queries uses three-valued logic for reasoning. Evaluation algorithms of a similar nature have been explored in the knowledge base literature [41], sometimes even using database evaluation techniques [42]. While not directly applicable to relational databases, the connection is worth studying, especially for representing certain answers as knowledge.

Easy to understand representation mechanisms for query answers. Our results suggest that it may be easier – and perhaps more natural sometimes – to produce certain answers in the form of knowledge rather than objects. This of course has been explored in the database literature. One of the early ideas was to use minimal disjunctive answers, cf. [43], incorporating disjunctions into query answers. An analog of those in the order-based setting is finding not the greatest lower bound of a set X but rather a finite set X_0 such that every element of X is above some element of X_0. A more expressive mechanism is that of conditional tables. Already in [2] it was shown that under CWA they form a strong representation system. Conditional tables, however, as explained earlier, are essentially FO formulae, from a rather bizarre fragment, that are simply made to look a bit like tables. Thus, mechanisms like disjunctive answers are preferable as they are much easier to understand, but we are still very far from having a good understanding of the expressiveness of various representation mechanisms.

We now give a couple of examples showing that simple representation mechanisms can be quite useful, and an alternative to hard-to-understand conditional tables. Take a query $Q = R - S$, and a database D with $R^D = \{1, 2\}$ and $S^D = \{3, \bot\}$. Under CWA, we have $Q([\![D]\!]_{\text{CWA}}) = \{\ \{1, 2\}, \{1\}, \{2\}\ \}$. If we take the greatest lower bound of these, then for $\preceq_{\text{CWA}}$ it simply does not exist, and for $\preceq_{\text{OWA}}$ ordering, it is $\emptyset$, which is, incidentally, what a commercial DBMS would return if one runs this query in SQL, written as `select r.a from r where r.a not in (select * from s)`. 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 we get essentially minimal disjunctive answers, as $Q([\![D]\!]_{\text{CWA}})$ can be described by $A(1) \vee A(2)$. If we consider the same query under OWA, then $Q([\![D]\!]_{\text{OWA}}) = \{A \mid 3 \notin A\}$. Again, this is not representable by a single object: taking the greatest lower bound of $Q([\![D]\!]_{\text{OWA}})$ 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. This suggests that one needs a deeper study of certain answers as knowledge, concentrating on representations of answers that make sense to users.

Applications. We explained that certain answers are the preferred way of dealing with incompleteness in many applications, including data integration, data exchange, ontology based query answering, and consistent query answering. So far we have used the approach proposed here to understand query answering over incomplete relational databases (see Section 7.2 and [38]). Our next goal is to use the same ideas to understand when we can make answering queries in the application scenarios easier. In some cases, query answering in those scenarios is reduced to building a database (typically incomplete) and answering queries over it (e.g., in data exchange). Then we can directly take advantage of techniques developed here. One question is to identify cases when this can be done.

In other cases, most likely we would need to use Proposition 8 as the starting point, and look for approximations. Note also that approximations are easier to find under CWA rather than OWA, as is already evident in the case of relational queries [38]. Thus, using our approach to find good quality and computationally efficient answers may be more challenging in settings relying on OWA (like ontology based query answering), although incorporating closed-world reasoning [44] can perhaps make finding approximate answers easier.

Other data models. Yet another direction is to apply the framework to non-relational models, particularly semi-structured and XML, for which incompleteness has been studied extensively [45, 26, 46, 31], and beyond, to graph data and RDF, where only preliminary results have been established so far [47, 48].

Acknowledgments. I would like to thank Pablo Barceló, Marco Console, Paolo Guagliardo, and Cristina Sirangelo for many discussions of incomplete information. I am grateful to the anonymous referees for their very careful reading of the paper and numerous suggestions for improving the presentation. I am also grateful to the referees of the KR 2014 paper whose comments are reflected in this full version. Work partly supported by EPSRC grants J015377 and M025268.

References

[1] S. Abiteboul, P. Kanellakis, G. Grahne, On the representation and querying of sets of possible worlds., Theoretical Computer Science 78 (1) (1991) 158–187.
[2] T. Imielinski, W. Lipski, Incomplete information in relational databases, Journal of the ACM 31 (4) (1984) 761–791.
[3] R. Reiter, Towards a logical reconstruction of relational database theory, in: On Conceptual Modelling, 1982, pp. 191–233.
[4] M. Lenzerini, Type data bases with incomplete information, Inf. Sci. 53 (1-2) (1991) 61–87.
[5] M. Lenzerini, Data integration: a theoretical perspective, in: ACM Symposium on Principles of Database Systems (PODS), 2002, pp. 233–246.
[6] M. Arenas, P. Barceló, L. Libkin, F. Murlak, Foundations of Data Exchange, Cambridge University Press, 2014.
[7] A. Calì, G. Gottlob, T. Lukasiewicz, A general datalog-based framework for tractable query answering over ontologies, J. Web Sem. 14 (2012) 57–83.
[8] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family, J. Autom. Reasoning 39 (3) (2007) 385–429.
[9] L. Bertossi, Database Repairing and Consistent Query Answering, Morgan&Claypool Publishers, 2011.
[10] D. Suciu, D. Olteanu, C. Re, C. Koch, Probabilistic Databases, Morgan&Claypool Publishers, 2011.
[11] W. Fan, F. Geerts, Foundations of Data Quality Management, Morgan&Claypool Publishers, 2012.
[12] J. Grant, Null values in a relational data base, Inf. Process. Lett. 6 (5) (1977) 156–157.
[13] W. Lipski, On semantic issues connected with incomplete information databases, ACM Transactions on Database Systems 4 (3) (1979) 262–296.
[14] S. Abiteboul, O. Duschka, Complexity of answering queries using materialized views, in: Proceedings of the 17th ACM Symposium on Principles of Database Systems, PODS'98, 1998, pp. 254–263.
[15] P. Barceló, Logical foundations of relational data exchange, SIGMOD Record 38 (1) (2009) 49–58.
[16] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, M. Zakharyaschev, The combined approach to ontology-based data access, in: IJCAI, 2011, pp. 2656–2661.
[17] P. Buneman, A. Jung, A. Ohori, Using powerdomains to generalize relational databases, Theoretical Computer Science 91 (1) (1991) 23–55.
[18] L. Libkin, A semantics-based approach to design of query languages for partial information, in: Semantics in Databases, Vol. 1358 of Lecture Notes in Computer Science, Springer-Verlag, 1995, pp. 170–208.
[19] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.
[20] R. Reiter, Equality and domain closure in first-order databases, Journal of the ACM 27 (2) (1980) 235–249.
[21] A. Gheerbrant, L. Libkin, C. Sirangelo, Naïve evaluation of queries over incomplete databases, ACM Trans. Database Syst. 39 (4) (2014) 31:1–31:42.
[22] C. J. Date, Database in Depth - Relational Theory for Practitioners, O'Reilly, 2005.
[23] C. J. Date, A critique of Claude Rubinson's paper 'Nulls, three-valued logic, and ambiguity in SQL: critiquing Date's critique', SIGMOD Record 37 (3) (2008) 20–22.
[24] C. J. Date, H. Darwen, A Guide to the SQL Standard, Addison-Wesley, 1996.
[25] M. Arenas, L. Libkin, XML data exchange: Consistency and query answering, Journal of the ACM 55 (2) (2008) 7:1–7:65.
[26] P. Barceló, L. Libkin, A. Poggi, C. Sirangelo, XML with incomplete information, Journal of the ACM 58 (1) (2010) 4:1–4:63.
[27] A. Gheerbrant, L. Libkin, T. Tan, On the complexity of query answering over incomplete XML documents, in: International Conference on Database Theory (ICDT), 2012, pp. 169–181.
[28] J. Barwise, S. Feferman (Eds.), Model-Theoretic Logics, Springer Verlag, 1985.
[29] C. Gunter, Semantics of Programming Languages: Structures and Techniques, MIT Press, 1992.

[30] L. Libkin, Incomplete information and certain answers in general data models, in: ACM Symposium on Principles of Database Systems (PODS), 2011, pp. 59–70.
[31] C. David, L. Libkin, F. Murlak, Certain answers for XML queries, in: ACM Symposium on Principles of Database Systems (PODS), 2010, pp. 191–202.
[32] K. Compton, Some useful preservation theorems, Journal of Symbolic Logic 48 (2) (1983) 427–440.
[33] B. Rossman, Homomorphism preservation theorems, Journal of the ACM 55 (3) (2008) 15:1–15:54.
[34] C. Chang, H. Keisler, Model Theory, North Holland, 1990.
[35] A. K. Chandra, D. Harel, Computable queries for relational data bases, J. Comput. Syst. Sci. 21 (2) (1980) 156–178.
[36] M. Arenas, P. Barceló, J. Reutter, Query languages for data exchange: beyond unions of conjunctive queries, in: International Conference on Database Theory (ICDT), 2009, pp. 73–83.
[37] R. Fagin, Inverting schema mappings, ACM Transactions on Database Systems 32 (4) (2007) 25:1–25:53.
[38] L. Libkin, SQL's three-valued logic and certain answers, in: Proceedings of the 18th International Conference on Database Theory (ICDT), 2015, pp. 94–109, full version to appear in *ACM Transactions on Database Systems*.
[39] R. Ramakrishnan, J. Gehrke, Database Management Systems, McGraw-Hill, 2003.
[40] R. Reiter, A sound and sometimes complete query evaluation algorithm for relational databases with null values, Journal of the ACM 33 (2) (1986) 349–370.
[41] H. J. Levesque, A completeness result for reasoning with incomplete first-order knowledge bases, in: KR, 1998, pp. 14–23.
[42] Y. Liu, H. J. Levesque, A tractability result for reasoning with incomplete first-order knowledge bases, in: IJCAI, 2003, pp. 83–88.
[43] R. van der Meyden, Logical approaches to incomplete information: A survey, in: Logics for Databases and Information Systems, 1998, pp. 307–356.
[44] C. Lutz, I. Seylan, F. Wolter, Ontology-mediated queries with closed predicates, in: International Joint Conference on Artificial Intelligence, 2015, pp. 3120–3126.
[45] S. Abiteboul, L. Segoufin, V. Vianu, Representing and querying XML with incomplete information, ACM Transactions on Database Systems 31 (1) (2006) 208–254.
[46] D. Calvanese, G. De Giacomo, M. Lenzerini, Semi-structured data with constraints and incomplete information, in: Description Logics, 1998.
[47] P. Barceló, L. Libkin, J. Reutter, Querying regular graph patterns, Journal of the ACM 61 (1) (2014) 8:1–8:54.
[48] C. Nikolaou, M. Koubarakis, Incomplete information in RDF, in: RR, 2013, pp. 138–152.