# Negative Knowledge for Certain Query Answers

Leonid Libkin[1]

School of Informatics, University of Edinburgh

**Abstract.** Querying incomplete data usually amounts to finding answers we are certain about. Standard approaches concentrate on positive information about query answers, and miss negative knowledge, which can be useful for two reasons. First, sometimes it is the only type of knowledge one can infer with certainty, and second, it may help one find good and efficient approximations of positive certain answers. Our goal is to consider a framework for defining both positive and negative certain knowledge about query answers and to show two applications of it. First, we demonstrate that it naturally leads to a way of representing certain information that has hitherto not been used in querying incomplete databases. Second, we show that approximations of such certain information can be computed efficiently for all first-order queries over relational databases.

## 1 Introduction

If uncertainty occurs in a dataset, answering queries against it typically involves computing *certain answers*, i.e., answers one can be sure about. This happens in traditional database query answering [2, 22] and in numerous applications such as data integration [26], data exchange [4], inconsistent databases [7], and ontology-based data access [11, 25]. The most common approach is to look at all complete datasets $D'$ that can potentially represent an incomplete dataset $D$ – i.e., its *semantics* $[\![D]\!]$ – and answers that are true in all such $D'$. When a query $Q$ returns sets of objects (for example, sets of tuples for relational database queries), certainty is typically defined by $\mathsf{certain}(Q, D) = \bigcap\{Q(D') \mid D' \in [\![D]\!]\}$, see [30]. This definition has been so dominant in the literature that even in models where queries do not return sets, languages have been adjusted to make this definition applicable (e.g., for XML and graph data [3, 5, 6]).

Certain answers defined this way can be viewed as a variant of the logical validity problem. This, not surprisingly, leads to high complexity bounds; in fact, query answering tends to be tractable for conjunctive queries or relatives, but computationally infeasible beyond [1, 4, 5, 7, 8, 10, 26, 35]. A very common situation is that adding features to conjunctive queries or their unions makes finding certain answers coNP-hard or even undecidable. It is thus well understood that the inability of the standard theoretical solutions to handle the problem of querying incomplete data outside a limited class of queries needs to be addressed. Recently, two lines of work in this direction have been pursued. The first revisits the very notion of certainty in query answering, and the second attempts to approximate certain answers efficiently.

The first line of works in fact dates back to the 1980s, when an alternative (and, as several papers [24, 28] have argued, better) definition of certain answers appeared

[31]. More recently, a general and data model-independent approach to defining query answers over incomplete databases was proposed in [28]. It was based on combining classical data management techniques with viewing databases as logical theories, as advocated by [33, 34], as well as using the idea of ordering incomplete databases in terms of their informativeness [9]. Certain answers can be represented by logical formulae true about answers in all possible worlds, and the notion of certainty is closely connected to logical entailment, rather than an arbitrary choice of intersection in the definition of certain. Using informativeness ordering, one can state when a query answering algorithm behaves rationally: this happens if it produces more informative answers on more informative inputs. For relational databases, these ideas led to new large classes of queries for which certain answers can be computed efficiently [17], and to a new account of many-valued query answers [13], as employed by all standard DBMSs [14].

The second line of work, based on approximations, was also used recently to show that an efficient approximation of certain answers can be computed for all first-order queries [29], not just unions of conjunctive queries, as was previously known [22]. A crucial element of that approach is that one needs to carry *negative certain* information while computing the answer, although at the end such negative information is dismissed and only the positive answer is given to the user.

However, dismissing negative information is not always a good path to follow, as it may in fact provide us with useful information about query answers. For example, consider a database $D$ with two unary relations $R$ and $S$, so that $R$ contains an unknown value (a *null* in the database terminology), $S = \{1\}$, and the query $Q(x) = R(x) \wedge \neg S(x)$ computes their difference. Then the certain answer is empty under every reasonable semantics. But we can be certain that 1 is not in the answer; hence, we are certain about the fact $\neg A(1)$ (with $A$ for "answer"), which says that while we do not know what may occur in the output, we do know that $1$ does not occur. Even though in this example $A(1)$ is the certain answer for the negation of $Q$, in general certain negative answers to $Q$ are not the same as what is known with certainty about $\neg Q$. Indeed, consider relations $R' = \{(1, \bot)\}$ and $S' = \{(1, \bot')\}$, where $\bot, \bot'$ indicate nulls (not necessarily denoting the same value). The negation of $Q'(\bar{x}) = R'(\bar{x}) \wedge \neg S'(\bar{x})$ is $S'(\bar{x}) \vee \neg R'(\bar{x})$, and thus, with certainty, the answer to $\neg Q'$ will have a tuple whose first component is 1, i.e., we know $\exists y A(1, y)$ about the answer to $Q'$. However, we cannot tell which tuples with certainty do *not* belong to the answer to $Q'$.

Even these simple examples tell us that the user may benefit from having negative certain information about query answers, and getting it involves more than just finding certain answers for the negation of the query. To understand how such negative information can be incorporated into query answering, we need to address several questions:

(a) How do we define negative and positive knowledge about query answers, and what is the connection between the two?
(b) In what logical languages can we express such negative knowledge? Can such knowledge be represented in a user-friendly way, and if so, does it correspond to any of the known ways of defining query answers?
(c) What is the complexity of finding positive and negative knowledge about query answers?

(d) If exact computation is infeasible, can we effectively approximate answers with some guarantees?

To answer the first question, we follow the approach of [17, 28] which treats incompleteness at an abstract level applicable to many data models. The key elements of the approach are the notions of complete and incomplete models, the *semantics* of an incomplete object, which is a set of complete ones it can denote, and a set of formulae representing *knowledge* about objects. The semantic function makes it possible to define *informativeness ordering*, which says when one object is more informative than another. The restriction of frameworks in [17, 28] was that knowledge was *positive*: if a fact is known about an object, it remains true in more informative ones. Negative knowledge is not such: we can think of it as saying that we do not know some fact about an object; therefore, we do not know that fact about less informative objects.

Positive formulae were used in [28] to define certain knowledge about sets of objects, providing a disciplined notion of certain answers, rather than an ad hoc one based on the notion of intersection. The idea is as follows: the *theory* of a set of objects is everything we know about that set with certainty. Such a theory of course could be infinite, but if we find a single formula equivalent to it, then this formula gives us a proper representation of certain knowledge.

To see what kinds of formulae we can use for negative knowledge, we follow a similar approach, but conditions required for good behavior of negative knowledge impose significant computational requirements, despite a seemingly simple reversal of the ordering. But we turn this to our advantage and use such conditions as a guide for finding logical formalisms for negative formulae. For relational databases, this results in a new formalism that exhibits a *duality* between formulae and objects, making it possible to apply effective query evaluation to compute certain knowledge.

This new formalism for defining certain answers (both positive and negative) is closely related to standard approaches used in the literature [22, 31] and yet is not covered by them. In essence, it allows nulls from the input database to be present in query answers (which is more than [22] does) but only allows repetitions of such nulls within a single tuple (as opposed to [31], which allows repetitions across different tuples in the answer).

To demonstrate the usefulness of this approach and the new representation mechanism for relational databases, we show how to compute both positive and negative knowledge about certain answers for all first-order (equivalently, relational algebra/calculus) queries over relational databases. Given the intractability of certain answers even for Boolean first-order queries [1], our procedure gives an approximation for those, which is efficient, and comes with correctness guarantees.

**Organization** Background material is presented in Section 2. Modeling negative knowledge is described in Section 3, and certain negative knowledge is studied in Section 4. Section 5 explains how to represent such knowledge for relational databases, and in Section 6 we provide an efficient algorithm for computing it.

## 2 Preliminaries

**A general model** We now recall the basic setting of [17, 28] that lets us talk about the essential features of incompleteness without recourse to a particular data model. The two basic concepts are *objects*, and *formulae* they satisfy. Objects could be incomplete or complete; the semantics of an incomplete object is the set of complete objects it may represent.

Formally, a *database domain* is a triple $\mathbb{D} = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!] \rangle$, where $\mathcal{D}$ is a set of objects (for instance, all relational databases over the same schema), $\mathcal{C}$ is the set of complete objects (for instance, databases over the same schema without incomplete information), and $[\![\,]\!] : \mathcal{D} \to 2^{\mathcal{C}}$ is the semantic function: $[\![x]\!] \subseteq \mathcal{C}$ is the semantics of an object $x$. We require that a complete object denote at least itself: if $c \in \mathcal{C}$, then $c \in [\![c]\!]$.

The *information ordering* is defined by

$$x \preceq y \;\Leftrightarrow\; [\![y]\!] \subseteq [\![x]\!]. \tag{1}$$

That is, the more an object denotes, the less we know about it (indeed, if we know nothing about something, it can denote everything). We require that objects in the semantics of $x$ be at least as informative as $x$: if $c \in [\![x]\!]$, then $x \preceq c$. This condition holds for all the standard semantics of incompleteness.

We also assume that we have a set of formulae $\mathbb{F}$ that express knowledge about objects in $\mathcal{D}$ and a satisfaction relation $\models$ between $\mathcal{D}$ and $\mathbb{F}$; that is, $x \models \varphi$ if $\varphi$ is true in $x$. For sets of objects and formulae, we write $X \models \varphi$ if $x \models \varphi$ for each $x \in X$, and $x \models \Phi$ if $x \models \varphi$ for each $\varphi \in \Phi$. As usual, $\mathsf{Th}(X) = \{\varphi \mid X \models \varphi\}$ is the *theory* of $X$, and $\mathsf{Mod}(\Phi) = \{x \mid x \models \Phi\}$ is the set of models of $\Phi$.

Previously, only *positive* knowledge was considered, i.e., it was required that $x \preceq y$ and $x \models \varphi$ imply $y \models \varphi$.

For domains $\mathbb{D} = \langle \mathcal{D}, \mathcal{C}, [\![\,]\!] \rangle$ and $\mathbb{D}' = \langle \mathcal{D}', \mathcal{C}', [\![\,]\!]' \rangle$, a *query* is modeled as a mapping $Q : \mathcal{D} \to \mathcal{D}'$ such that $Q(c) \in \mathcal{C}'$ whenever $c \in \mathcal{C}$ (no incompleteness is introduced when a query acts on a complete object). Note that the semantics $[\![\,]\!]'$ of query *answers* need not be the same as the semantics $[\![\,]\!]$ of query inputs.

The main object one then works with [2, 22] is

$$Q([\![x]\!]) \;=\; \{Q(c) \mid c \in [\![x]\!]\} \;\subseteq\; \mathcal{D}' \tag{2}$$

which gives us the answers to $Q$ in all possible worlds representing $x$. Finding certain answers to $Q$ on $x$ then amounts to extracting what we know with certainty about $Q([\![x]\!])$.

**Certain knowledge** Since computing certain answers amounts to extracting certain information from a set of objects, typically of the form (2), we need to know how to describe certain information in a set $X \subseteq \mathcal{D}$. We know that $\mathsf{Th}(X)$ is the set of facts that are true in all objects of $X$, i.e., this is what we know about $X$ with certainty. The whole theory is not an object we want to work with for performing computational tasks (to start with, it is likely to be infinite). What we want instead is a single formula equivalent to this theory; then such a formula describes all the certain knowledge of

$X$. Of course formulae/theories are equivalent when they have the same models. Using this, [28] proposed to define certain knowledge of a set of objects as a formula $\Box X$ such that

$$\mathsf{Mod}(\Box X) \;=\; \mathsf{Mod}(\mathsf{Th}(X)) \tag{3}$$

Such a formula may not exist for all sets $X$ (by a simple cardinality argument), although in many cases relevant for query answering, it does. It need not be unique, but this is not a problem: if both $\mathsf{Mod}(\varphi_1)$ and $\mathsf{Mod}(\varphi_2)$ equal $\mathsf{Mod}(\mathsf{Th}(X))$, then $\varphi_1$ and $\varphi_2$ are equivalent, as formulae having the same models, and hence either one can be used as $\Box X$.

**Incomplete relational databases** As a concrete example of incomplete information, we consider relational databases with naïve, or marked nulls [2, 22]. This model dominates in applications such as exchange and integration of data [26, 4], and subsumes the usual model of nulls implemented in commercial DBMSs. In this model, there are two types of values: *constants* and *nulls*. There are countably infinite sets Const of constants (e.g., $1, 2, \ldots$), and Null of nulls, which will be denoted by $\bot$, with sub- or superscripts.

A relational *vocabulary* (or *schema*) is a set of relation names, each with its arity. An incomplete relational database $D$ associates with each $k$-ary relation symbol $R$ from the vocabulary a $k$-ary relation $R^D \subseteq (\mathsf{Const} \cup \mathsf{Null})^k$. When $D$ is clear from the context, we write $R$ rather than $R^D$. 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 $\text{adom}(D) = \mathsf{Const}(D) \cup \mathsf{Null}(D)$. A *complete* database $D$ has no nulls, i.e., $\text{adom}(D) \subseteq \mathsf{Const}$.

The basic semantics of incomplete databases is given by means of special kinds of homomorphisms between instances. A map $h : \mathsf{Null} \to \mathsf{Const} \cup \mathsf{Null}$ is a *homomorphism* between two instances $D$ and $D'$ if for each relation symbol $R$, if $\bar{t} \in R^D$, then $h(\bar{t}) \in R^{D'}$. Here $h(v_1, \ldots, v_k) = (h(v_1), \ldots, h(v_k))$, and we assume that $h(v) = v$ for each $a \in \mathsf{Const}$.

A homomorphism is called a *valuation* if $h(v) \in \mathsf{Const}$ for each $v$. By $h(D)$ we denote the image of a homomorphism, i.e., the database consisting of all the tuples $h(\bar{t})$ for $\bar{t} \in R^D$, for each relation $R$ in the vocabulary.

The standard semantics of incompleteness [22] are the *closed world assumption* (CWA) and the *open world assumption* (OWA) semantics:

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

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

The former simply replaces nulls by constants, and the latter in addition allows us to add any set of complete tuples.

The information orderings (1) given by these semantics are as follows: for OWA, $D \preceq_{\text{OWA}} D'$ iff there is a homomorphism $h : D \to D'$, and for CWA, $D \preceq_{\text{CWA}} D'$ iff there is a homomorphism $h : D \to D'$ such that $D' = h(D)$, see [17].

**Queries** A *relational query* of arity $k$ maps databases $D$ over a relational schema into a single $k$-ary relation, which we denote here by $A$ (for 'answers'). This is in line with standard languages such as relational calculus, relational algebra, and SQL, whose queries specify attributes of an output table [2, 14].

The classical definition [22] of certain answers in the literature is the set $\mathsf{certain}(Q, D)$ of tuples $\bar{u}$ over $\mathsf{Const}$ such that $\bar{u} \in Q(D')$ for every $D' \in [\![D]\!]$. Note that answers depend on the semantics of the input. Another definition, which has the advantage of keeping nulls in answers, is that of *certain answers with nulls*, $\mathsf{certain}_\perp(Q, D)$ (it was first defined in [31] although not given a name; the name we use is from [29]). For CWA, the set $\mathsf{certain}_\perp(Q, D)$ consists of all tuples $\bar{u}$ over $\mathrm{adom}(D)$ – thus having both constants and nulls – such that for every valuation $h$ on $D$, we have $h(\bar{u}) \in Q(h(D))$. It turns out that, under CWA, $\mathsf{certain}(Q, D)$ is precisely the set of constant tuples in $\mathsf{certain}_\perp(Q, D)$.

For relational databases, as our basic language we consider *first-order logic* (FO) over the relational vocabulary (i.e., relational calculus, which also serves as the basis of SQL [2]). More precisely, its atomic formulae are relational atoms $R(\bar{x})$ and equality atoms $x = y$, and its formulae are closed under Boolean connectives $\wedge, \vee, \neg$ and quantifiers $\exists, \forall$. The $\exists, \wedge$-closure of atomic formulae is referred to as the set of *conjunctive queries*; those are of the form $\varphi(\bar{x}) = \exists \bar{z} \bigwedge_i R_i(\bar{z}_i)$ where each $R_i$ is a relation symbol and variables in tuple $\bar{z}_i$s come from $\bar{x}$ and $\bar{y}$.

## 3 Modeling negative knowledge

So far we assumed that the knowledge of objects is positive: a formula $\varphi$ true in an object continues to be true when an object is replaced by a more informative one. While in general we often deal with logical formalisms not closed under negation (e.g., conjunctive queries), assume for now that we can negate $\varphi$. If $\neg\varphi$ is true an object $x$, and $y \preceq x$, then we would have $y \models \neg\varphi$. Thus, to model negative knowledge in general, we look at formulae whose sets of models are downward closed. In other words, we now have two sets of formulae, $\mathbb{F}^+$ and $\mathbb{F}^-$, such that,

- for $\varphi \in \mathbb{F}^+$, if $x \preceq y$ and $x \models \varphi$, then $y \models \varphi$;
- for $\psi \in \mathbb{F}^-$, if $x \preceq y$ and $y \models \psi$, then $x \models \psi$.

There appear to be two possible approaches to extending the framework of [28] with both certain positive and certain negative knowledge.

*The first approach* We follow the idea behind the definition (3). We can define $\mathsf{Th}^+(X) = \{\varphi \in \mathbb{F}^+ \mid X \models \varphi\}$ and $\mathsf{Th}^-(X) = \{\psi \in \mathbb{F}^- \mid X \models \psi\}$ as theories expressing positive and negative knowledge about $X$, and then, as in (3), try to capture them with formulae $\Box^+ X$ and $\Box^- X$ such that

$$\begin{array}{c}\mathsf{Mod}(\Box^+ X) = \mathsf{Mod}(\mathsf{Th}^+(X)) \\ \mathsf{Mod}(\Box^- X) = \mathsf{Mod}(\mathsf{Th}^-(X))\end{array} \tag{4}$$

When they exist, these formulae represent certain positive knowledge and certain negative knowledge about $X$. Note that $\mathsf{Mod}(\Box^+ X)$ is upward-closed and $\mathsf{Mod}(\Box^- X)$ is downward-closed with respect to $\preceq$.

*The second approach* Note that (3) is based on an equivalence between two theories: $\Phi \approx_{\mathsf{tt}} \Psi$ whenever for each object $x$, all formulae of $\Phi$ are true in $x$ iff all formulae of $\Phi$ are true in $x$. Then we just required that $\Box^+ X \approx_{\mathsf{tt}} \mathsf{Th}^+(X)$.

An alternative is to look at equivalence with respect to negative information, essentially changing true and false. We let $\Phi \approx_{\mathsf{ff}} \Psi$ whenever for each object $x$, all formulae of $\Phi$ are false in $x$ iff all formulae of $\Psi$ are false in $x$. It would make sense then to capture all things we know to be false in $X$ using this equivalence. That is, we define

$$\mathsf{Th}^+_\neg(X) = \{\varphi \in \mathbb{F}^+ \mid X \models \neg\varphi\}$$
$$\mathsf{Th}^-_\neg(X) = \{\psi \in \mathbb{F}^- \mid X \models \neg\psi\}$$

as sets of formulae we know with certainty are false in $X$, and then try to capture them with single formulae satisfying

$$\Box^+_\neg X \approx_{\mathsf{ff}} \mathsf{Th}^+_\neg(X) \;\text{ and }\; \Box^-_\neg X \approx_{\mathsf{ff}} \mathsf{Th}^-_\neg(X). \tag{5}$$

Both of these seem to be reasonable ways of capturing negative information about a set of objects; fortunately, they are closely related so we can choose either (4) or (5) as the main definition. For formulae $\alpha, \beta$, we write $\alpha = \neg\beta$ if $\mathsf{Mod}(\alpha) = \mathcal{D} - \mathsf{Mod}(\beta)$ (so $\alpha = \neg\beta$ implies $\beta = \neg\alpha$).

**Theorem 1.** *Assume that $\mathbb{F}^-$ contains exactly the negations of formulae in $\mathbb{F}^+$. If formulae $\Box^* X$ and $\Box^*_\neg X$ exist, when $*$ is $+$ or $-$, we have the following relationships between them: $\Box^+ X = \neg\Box^-_\neg X$ and $\Box^- X = \neg\Box^+_\neg X$.*

To illustrate the difference between two ways of representing negative information, consider a database with relations $R = \{\bot\}$ and $S = \{1, 2\}$, and a query $Q$ that computes their difference $R - S$, i.e., $Q(x) = R(x) \land \neg S(x)$. Let $X = Q([\![R, S]\!])$ under either OWA or CWA, and consider $\mathbb{F}^+$ that consists of atomic relational formulae. Then $\mathsf{Th}^+_\neg(X)$ contains $A(1)$ and $A(2)$, and thus $\Box^+_\neg X$ is equivalent to $A(1) \lor A(2)$. That is, $\Box^+_\neg X$ describes what we know with certainty will *not* hold in the answer to the query. On the other hand, $\Box^- X$ is equivalent to $\neg A(1) \land \neg A(2)$ (again, assuming the connection between $\mathbb{F}^+$ and $\mathbb{F}^-$ as in the theorem) and describes negative information that is guaranteed to be true in the query result.

## Certain knowledge for query answering

Given an object $x$ and a query $Q$, answering $Q$ on $x$ in a way that provides both positive and negative knowledge amounts to finding the pair of formulae

$$\Box(Q, x) \;=\; \big(\,\Box^+ Q([\![x]\!]),\; \Box^- Q([\![x]\!])\,\big) \tag{6}$$

whenever such formulae exist, and their computation is feasible. For representing the second component, we can choose either $\Box^- Q([\![x]\!]))$, or its negation $\Box^+_\neg Q([\![x]\!])$, as Theorem 1 suggests. The components of (6) are the most general formulae defining positive and negative knowledge, as they imply all formulae in $\mathsf{Th}^+(Q([\![x]\!]))$ and $\mathsf{Th}^-(Q([\![x]\!]))$, respectively. If computing them is infeasible, we can look for *approximations* by means of returning a pair $(\alpha, \beta)$ of formulae such that $\alpha \in \mathsf{Th}^+(Q([\![x]\!]))$ and $\beta \in \mathsf{Th}^-(Q([\![x]\!]))$. They may not be as general as (6), but they still give us information about query answers we can be certain about.

## 4 Certain negative knowledge

Our next goal is to understand how to represent certain negative and positive information, particularly for sets $X$ which are possible query answers, as in (2). That is, we will see what requirements must be imposed on logical formalisms $\mathbb{F}^+$ and $\mathbb{F}^-$ to ensure feasible computation of certain answers.

Towards understanding these requirements, we present an alternative view of formulae $\Box^+X$ and $\Box^-X$. For that, consider the usual implication of formulae, $\varphi \supset \psi$ iff $\mathsf{Mod}(\varphi) \subseteq \mathsf{Mod}(\psi)$. It generates a preorder (reflexive transitive relation) on sets $\mathbb{F}^+$ and $\mathbb{F}^-$. Viewing implication as a preorder, we define, for a set of formulae $\Phi$, the formula $\bigwedge \Phi$ as the *greatest lower bound* in the preorder $\supset$. That is, $\bigwedge \Phi \supset \Phi$ and whenever $\varphi' \supset \Phi$, we have that $\varphi' \supset \bigwedge \Phi$ (here $\varphi' \supset \Phi$ means that $\varphi'$ implies every formula $\varphi \in \Phi$). These formulae let us capture certain knowledge provided by $\Phi$, so it seems desirable to have $\Box^*X$ to be the same as $\bigwedge \mathsf{Th}^*(X)$, for $*$ being $+$ or $-$. We now explain when this is possible.

First, we remark that formulae $\bigwedge \Phi$ may not exist in general, and if they exist, they may not be unique, although any two such formulae are logically equivalent since they have the same models. While the notation $\bigwedge$ is standard for greatest lower bounds, the connection with conjunction is natural: if there is a formula $\varphi$ equivalent to the conjunction of all formulae in $\Phi$, then $\mathsf{Mod}(\varphi) = \mathsf{Mod}(\Phi)$ and $\varphi = \bigwedge \Phi$; in general though we may have $\mathsf{Mod}(\bigwedge \Phi) \subsetneq \mathsf{Mod}(\Phi)$.

We now show when $\Box^*X = \bigwedge \mathsf{Th}^*(X)$. In fact, for $\mathsf{Th}^+$, this was already shown in [28], but under additional conditions that we now eliminate.

Let $\uparrow x = \{y \mid x \preceq y\}$ and $\downarrow x = \{y \mid y \preceq x\}$. By $\delta_x^{\uparrow}$ and $\delta_x^{\downarrow}$ we denote formulae (if they exist) such that $\mathsf{Mod}(\delta_x^{\uparrow}) = \uparrow x$ and $\mathsf{Mod}(\delta_x^{\downarrow}) = \downarrow x$.

**Theorem 2.**
– *If $\mathbb{F}^+$ is closed under conjunction and contains formulae $\delta_x^{\uparrow}$ for all $x$, then $\Box^+X = \bigwedge \mathsf{Th}^+(X)$ for every $X$.*
– *If $\mathbb{F}^-$ is closed under disjunction and contains formulae $\delta_x^{\downarrow}$ for all $x$, then $\Box^-X = \bigwedge \mathsf{Th}^-(X)$ for every $X$.*

The meaning of the equalities $\Box^*X = \bigwedge \mathsf{Th}^*(X)$ is that if one formula exists, then so does the other, and the two are equivalent, i.e., have the same models.

For most common semantics of incompleteness, formulae $\delta_x^{\uparrow}$ are easy to construct, and in fact they determine the shape of queries that can be answered easily under those semantics [17]. For instance, under OWA, they are conjunctive queries, and for CWA, they extend positive FO formulae with a limited form of guarded negation [12]. The new condition for $\mathsf{Th}^-$ that formulae $\delta_x^{\downarrow}$ be definable is harder to achieve, and this condition will let us choose the appropriate logical language for $\mathbb{F}^-$.

## 5 Representation of relational query answers: incomplete tuples

We now use the abstract results of two previous sections to suggest a representation mechanism for relational query answers, and to show how to find positive and negative answers using such a representation. For finding a representation mechanism, we analyze computational properties of formulae $\delta_x^{\uparrow}$ and $\delta_x^{\downarrow}$. Restricting those to a tractable class, gives us a representation of answers, called *incomplete tuples*. This representation exhibits a *duality* between formulae and objects: that is, positive and negative theories of query answers can be viewed as set of conventional tuples that use null values. With this duality, we define query answers using (6). To check that the definition makes sense, we have to make sure that it preserves informativeness. This, in turn, means that we need to define orderings on query answers, i.e., sets of incomplete tuples. We do so, and then prove, in Theorem 3 that the resulting representation mechanism and query answering by means of (6) do behave rationally, i.e., preserve informativeness.

We start by looking at the requirements of Theorem 2 and analyzing formulae $\delta_x^{\uparrow}$ and $\delta_x^{\downarrow}$. While the former are easy to obtain for standard semantics of incompleteness, the latter could become too expensive computationally, and it is their complexity that will suggest the representation of positive and negative certain answers.

We deal with relational databases, as described in Section 2. When we deal with outputs of relational queries, which are sets of tuples, it suffices to deal with one predicate for each type of answers, positive or negative (of course usual relational databases just return one set of tuples, for positive answers). As before, we refer to that predicate as $A(\cdot)$; later, when we look in more detail at separation of positive and negative answers, we shall use predicates $A^+(\cdot)$ and $A^-(\cdot)$.

The first observation shows that one must impose rather strong restriction on the types of formulae $\mathbb{F}^+$ that represent query answers (note that this does *not* imply any restriction on queries themselves).

**Proposition 1.** *For the class of conjunctive queries, data complexity of formulae $\delta_A^{\downarrow}$ for relations $A$ is in* NP*; in fact there is a relation for which data complexity of $\delta_A^{\downarrow}$ is* NP*-complete.*

Indeed, formulae $\delta_A^{\downarrow}$ test the existence of a homomorphism into $A$, i.e., they encode the general constraint satisfaction problem. In particular, such formulae are not expressible in FO, nor even its extensions with least and inflationary fixpoints.

**Incomplete tuples** The standard representation of query answers in relational databases is by means of ground tuples: one simply says that a tuple $\bar{a}$ is in the answer, or that predicate $A(\bar{a})$ holds. Proposition 1 says that extending it to conjunctive queries as a representation mechanism is too much from the complexity point of view. Over the vocabulary $A(\cdot)$ of query answers, Boolean conjunctive queries are of the form $\exists \bar{x}(A(\bar{x}_1, \bar{c}_1) \wedge \ldots \wedge A(\bar{x}_m, \bar{c}_m))$, where $\bar{c}_i$s are tuples of constants from $\mathsf{Const}$ and $\bar{x}_i$s are tuples of variables that together form $\bar{x}$. Eliminating variables gives us sets of constant tuples, i.e., the usual database query answers over complete data. Another way of simplifying the definition is to eliminate variables that occur in more than one $\bar{x}_i$, i.e., looking at formulae $\exists \bar{x}_1 A(\bar{x}_1, \bar{c}_1) \wedge \ldots \wedge \exists \bar{x}_m A(\bar{x}_m, \bar{c}_m)$. That is, we are dealing with conjunctions of formulae $\exists \bar{x} A(\bar{x}, \bar{c})$.

We can think of such formulae $\exists \bar{x} A(\bar{x}, \bar{c})$ as incomplete tuples. An *incomplete tuple* is simply a tuple of $\mathsf{Const} \cup \mathsf{Null}$. There is a natural correspondence between formulae $\exists \bar{x} A(\bar{x}, \bar{c})$ and incomplete tuples: for instance, $\exists x, x' A(x, 1, x, 2, x')$ can be thought of as an incomplete tuple $(\bot, 1, \bot, 2, \bot')$. Note that this duality between incomplete tuples as formulae and as actual tuples lets us represent query answers of the form (6) just as database relations.

Representation of answers by means of incomplete tuples is between the usual marked nulls and the Codd interpretation of nulls, which model SQL's view of nulls [2, 22]. Marked nulls can be repeated, and appear in different tuples; Codd nulls cannot be repeated at all. In incomplete tuples, a null can be repeated, but only within a tuple, and not across several tuples.

**Query answering and ordering**

We now consider orderings on query answers which are viewed as sets of incomplete tuples. Recall that we expect a rationally behaving query answering to produce more informative answers when more informative inputs are given; hence orderings are necessary to prove such rationality. For input databases, we have seen some standard orderings such as $\preceq_{\text{OWA}}$ and $\preceq_{\text{CWA}}$. According to (6), a query answer will be given as a pair of sets $(A^+, A^-)$ of incomplete tuples. Tuples in $A^+$ belong to the answer with certainty; thus, when viewed as formulae, their conjunction is equivalent to the formula $\Box^+ Q([\![x]\!])$. Tuples in $A^-$ are those that certainly do not belong to the answer; hence conjunction of their negations is equivalent to $\Box^- Q([\![x]\!])$.

First, we need to see how we can order incomplete tuples in terms of their informativeness. There are two ways of looking at it:

– What improves informativeness of a tuple? Replacing a null with a constant does, and replacing a null with another null might (e.g., if we replace $\bot'$ with $\bot$ in $(\bot, \bot')$, we get a more informative tuple $(\bot, \bot)$ giving extra information that its components are the same). Thus, given two incomplete tuples $\bar{a}$ and $\bar{b}$ over $\mathsf{Const} \cup \mathsf{Null}$, $\bar{b}$ is more informative than $\bar{a}$ if there is a homomorphism $h$ so that $h(\bar{a}) = \bar{b}$.
– Alternatively, we view incomplete tuples as formulae and say that $\bar{b}$ is more informative than $\bar{a}$ if it logically entails it, i.e., $\bar{b} \supset \bar{a}$.

The homomorphism theorem for conjunctive queries tell us that these two are equivalent, so we can take either of them as the definition of $\bar{b}$ being more informative than $\bar{a}$, which will be denoted by $\bar{a} \lhd \bar{b}$.

Next, we look at sets of incomplete tuples $A$ and $B$, and define orderings $\preceq^+_{\text{IT}}$ and $\preceq^-_{\text{IT}}$ saying that one of the sets has more positive or negative information than the other. First,

$$A \preceq^+_{\text{IT}} B \quad \Leftrightarrow \quad \forall \bar{a} \in A \; \exists \bar{b} \in B : \; \bar{a} \lhd \bar{b}.$$

This ordering says that we can improve an answer by improving individual tuples in it, or adding new tuples that our initial attempt to approximate query answers may have missed. This is the ordering on positive query answers we shall use. Note that it is also consistent with observations made in [28, 13] that for query answers (as opposed to inputs), the prefer interpretation is open-world, as adding tuples improves the answer.

When it comes to negative information, if we are given two incomplete tuples $\bar{a}$ and $\bar{b}$ such that $\bar{a} \lhd \bar{b}$, then it is actually better to have $\bar{a}$ in the answer, as it gives us more information about tuples to exclude. For instance, having a tuple $(1, 2)$ in the negative answer simply says that $(1,2)$ is never in the answer, but having a tuple $(1,\bot)$ is more informative as it says that no tuple whose first component is $1$ is in the answer. This leads to the following ordering:

$$A \preceq^-_{\text{IT}} B \quad\Leftrightarrow\quad \forall \bar{b} \in B\ \exists \bar{a} \in A:\ \bar{a} \lhd \bar{b}.$$

Note that these are well-known orderings in the semantics of concurrency (so called Hoare and Smyth powerdomain orderings [20]) where they are used to compare possible outcomes of different threads of concurrent computations in terms of the information they carry. In terms of computational problems, unlike the relations $\preceq_{\text{OWA}}$ and $\preceq_{\text{CWA}}$, we can test relations $\preceq^+_{\text{IT}}$ and $\preceq^-_{\text{IT}}$ in polynomial (quadratic) time.

A set $A$ of incomplete tuples can be viewed as a formula (which we also denote $A$, using the duality between tuples and formulae), which is the conjunction of all $\bar{a}$ in $A$. Likewise, we can also look at conjunction of all formulae $\neg\bar{a}$, giving us a formula $A^\neg$. That is, positive and negative formulae associated with $A$ are:

$$A \;=\; \bigwedge\{\bar{a} \mid \bar{a} \in A\} \qquad A^\neg \;=\; \bigwedge\{\neg\bar{a} \mid \bar{a} \in A\} \tag{7}$$

Note that the first equation simply extends the duality of incomplete tuples and formulae to sets of incomplete tuples: it just tells us how to view a set $A$ as a formula.

The following connection between orderings on sets and entailment of formulae is easily obtained from the definitions and containment criteria for conjunctive queries and their unions.

**Proposition 2.** *$A \preceq^+_{\text{IT}} B$ iff $B \supset A$, and $A \preceq^-_{\text{IT}} B$ iff $A^\neg \supset B^\neg$.*

Equipped with this, we can show that query answering by means of finding positive and negative incomplete tuples, i.e., by using (6), is always possible and preserves informativeness when input databases are interpreted under OWA or CWA.

**Theorem 3.** *Assume that input databases are interpreted under* OWA *or* CWA, *and that $\mathbb{F}^+$ consists of incomplete tuples, and $\mathbb{F}^-$ consists of their negations. Then for every query $Q$ and every database $D$ there exist finite sets $Q^+_\Box(D)$ and $Q^-_\Box(D)$ of incomplete tuples that, when viewed as formulae (7), are equivalent to $\Box^+ Q([\![D]\!])$ and $\Box^- Q([\![D]\!])$:*

$$\mathsf{Mod}(Q^+_\Box(D)) = \mathsf{Mod}(\mathsf{Th}^+(Q([\![D]\!])))$$
$$\mathsf{Mod}(Q^-_\Box(D)^\neg) = \mathsf{Mod}(\mathsf{Th}^-(Q([\![D]\!])))$$

*Moreover, this way of query answering preserves informativeness: if $D \preceq D'$ (under the ordering given by the* CWA *or the* OWA *semantics), then*

$$Q^+_\Box(D) \preceq^+_{\text{IT}} Q^+_\Box(D') \quad \textit{and} \quad Q^-_\Box(D) \preceq^-_{\text{IT}} Q^-_\Box(D').$$

## 6 Certain information via incomplete tuples

The conclusion of the previous section is that incomplete tuples are a good representational mechanism for query answers over incomplete relational databases. What makes them especially suitable for the task is the *duality* of incomplete tuples: each one can be viewed both as a formula $\exists\bar{x}A(\bar{a},\bar{x})$, satisfied by the query answer, or as an actual tuple $(\bar{x},\bar{a})$, where $\bar{x}$ is a tuple of nulls. Thus, a set of tuples can be seen both as sets of formulae (7) representing our knowledge (positive and negative) about query answers, and an actual database relation with nulls. This duality lets us compute such knowledge using well established database query evaluation techniques, and present it to the user in a familiar format.

Ideally, following Theorem 3, we want to compute, for a query $Q$ and a database $D$, sets $Q^+_\Box(D)$ and $Q^-_\Box(D)$ of incomplete tuples such that $Q^+_\Box(D)$ is equivalent to $\Box^+Q(\llbracket D\rrbracket)$ and $Q^-_\Box(D)^\neg$ is equivalent to $\Box^-Q(\llbracket D\rrbracket)$. That is,

$$\begin{aligned}\mathsf{Mod}(\textstyle\bigwedge\{\bar{a}\mid\bar{a}\in Q^+_\Box(D)\}) &= \mathsf{Mod}(\mathsf{Th}^+(Q(\llbracket D\rrbracket)))\ \text{ and}\\ \mathsf{Mod}(\textstyle\bigwedge\{\neg\bar{a}\mid\bar{a}\in Q^-_\Box(D)\}) &= \mathsf{Mod}(\mathsf{Th}^-(Q(\llbracket D\rrbracket)))\end{aligned}$$

This is problematic even for first-order queries, however, as computing such sets of incomplete tuples is expensive. In fact, a simple examination of proofs in [1, 18] shows that even when $Q$ is a fixed Boolean FO query, checking whether $\Box^+Q(\llbracket D\rrbracket_{\text{CWA}})$ is true is CONP-complete, and the same question for $\Box^+Q(\llbracket D\rrbracket_{\text{OWA}})$ is undecidable.

But the discussion following the definition (6) showed a way out of this problem: we need to compute approximate answers with some guarantees, that is, formulae from positive and negative theories of $Q(\llbracket D\rrbracket)$. Using the duality of incomplete tuples, we say that, for a query $Q$, the pair $(Q^+,Q^-)$ of queries returning sets of incomplete tuples provides a *sound answer* for $Q$ under $\llbracket\,\rrbracket$ if, for every database $D$,

$$Q^+(D)\subseteq\mathsf{Th}^+(Q(\llbracket D\rrbracket))\ \text{ and }\ Q^-(D)\subseteq\mathsf{Th}^+_\neg(Q(\llbracket D\rrbracket))\,. \tag{8}$$

Indeed, $Q^+(D)$ and $Q^-(D)$ represent parts of certain positive and negative knowledge about $Q(\llbracket D\rrbracket)$. If furthermore they can be computed with tractable data complexity, we say that they provide an *efficient sound answer* to $Q$ on $D$.

Note that the right way to read sound answers is *tuple-by-tuple*: for instance, if $(\bot,1)$ and $(\bot,2)$ are in $Q^+(D)$, the correct interpretation is that for every $D'\in\llbracket D\rrbracket$, the answer $Q(D')$ contains a tuple whose second component is 1, and a tuple whose second component is 2. It is not meant to say that the first components of such tuples are the same: incomplete tuples cannot make cross-tuple statements.

## Efficient sound answers under OWA and CWA

There are trivial ways of finding sound answers: for instance, by letting $Q^+$ and $Q^-$ return the empty set. Of course this is not what we want; instead we would like to find a good approximation of positive and negative certain information. To find the exact representation of such information, or a representation with some quality guarantees, and to do so efficiently, is impossible due to the complexity considerations explained earlier (which apply even to Boolean queries).

Thus, we shall present one particular inductive definition of queries $Q^+$ and $Q^-$ that provides efficient sound answers for the most commonly used semantics of incompleteness, i.e., OWA and CWA semantics, for all FO queries. We also assume, as is standard in the database context, that they are evaluated under the active domain semantics, i.e., the answer to a $k$-ary query $Q(\bar{x})$ on $D$, denoted by $Q(D)$, is the set of tuples

$\bar{a} \in \text{adom}(D)^k$ so that $D \models Q(\bar{a})$. Formulae $Q^+$ and $Q^-$ will use additional atomic formulae $\mathsf{const}(x)$ saying that $x$ is not a null, i.e., an element of $\mathsf{Const}$. We also write $\mathsf{const}(x_1, \ldots, x_n)$ for the conjunction of all $\mathsf{const}(x_i)$ for $1 \leq i \leq n$.

The definitions of $Q^+$ and $Q^-$ are identical for OWA and CWA, except in the case of relational atomic formulae. We now present them inductively for the following formulae constructors: $Q(\bar{x}, \bar{y}, \bar{z}) = Q_1(\bar{x}, \bar{y}) \wedge Q_2(\bar{x}, \bar{z})$ (to account properly for the use of free variables in conjuncts); $Q(\bar{x}, \bar{y}, \bar{z}) = Q_1(\bar{x}, \bar{y}) \vee Q_2(\bar{x}, \bar{z})$ (likewise for disjunction); $Q(\bar{x}) = \neg Q_1(\bar{x})$; and $Q(\bar{x}) = \exists y Q_1(\bar{x}, y)$; as well as equational atoms $x = y$ and $x = a$ for a constant $a \in \mathsf{Const}$.

– If $Q(\bar{x}, \bar{y}, \bar{z}) = Q_1(\bar{x}, \bar{y}) \wedge Q_2(\bar{x}, \bar{z})$, then

$$\begin{aligned} Q^+(\bar{x}, \bar{y}, \bar{z}) &= Q_1^+(\bar{x}, \bar{y}) \wedge Q_2^+(\bar{x}, \bar{z}) \wedge \mathsf{const}(\bar{x}) \\ Q^-(\bar{x}, \bar{y}, \bar{z}) &= Q_1^-(\bar{x}, \bar{y}) \vee Q_2^-(\bar{x}, \bar{z}) \end{aligned}$$

– If $Q(\bar{x}, \bar{y}, \bar{z}) = Q_1(\bar{x}, \bar{y}) \vee Q_2(\bar{x}, \bar{z})$, then

$$\begin{aligned} Q^+(\bar{x}, \bar{y}, \bar{z}) &= Q_1^+(\bar{x}, \bar{y}) \vee Q_2^+(\bar{x}, \bar{z}) \\ Q^-(\bar{x}, \bar{y}, \bar{z}) &= Q_1^-(\bar{x}, \bar{y}) \wedge Q_2^-(\bar{x}, \bar{z}) \end{aligned}$$

– If $Q(\bar{x}) = \neg Q_1(\bar{x})$, then $Q^+(\bar{x}) = Q_1^-(\bar{x})$ and $Q^-(\bar{x}) = Q_1^+(\bar{x}) \wedge \mathsf{const}(\bar{x})$.
– If $Q(\bar{x}) = \exists y Q_1(\bar{x}, y)$, then $Q^+(\bar{x}) = \exists y Q_1^+(\bar{x}, y)$ and $Q^-(\bar{x}) = \forall y Q_1^-(\bar{x}, y)$.
– If $Q(x) = (x = a)$, then $Q^+(x) = (x = a)$ and $Q^-(x) = \neg(x = a) \wedge \mathsf{const}(x)$.
– If $Q(x, y) = (x = y)$, then

$$Q^+(x, y) = (x = y) \text{ and } Q^-(x, y) = \neg(x = y) \wedge \mathsf{const}(x, y).$$

Note that the rules for $\wedge$ and $\vee$ are not symmetric, due to the asymmetric rule for negation.

Finally we define such queries for atomic formulae $R(\bar{x})$, when $R$ is a database relation, as follows:

| | | |
|---|---|---|
| Under OWA: | $R^+(\bar{x}) = R(\bar{x})$ | $R^-(x, y) = \mathsf{false}$ |
| Under CWA: | $R^+(\bar{x}) = R(\bar{x})$ | $R^-(x, y) = \neg \exists \bar{y} (R(\bar{y}) \wedge \alpha_{\lhd}(\bar{x}, \bar{y}))$ |

Here we use an additional formula $\alpha_{\lhd}(\bar{x}, \bar{y})$ such that $\alpha_{\lhd}(\bar{a}, \bar{b})$ iff $\bar{a} \lhd \bar{b}$. It is not hard to see that it can be defined as a quantifier-free formula that uses equalities and $\mathsf{const}(\cdot)$, as a disjunction over possible instantiations of variables $\bar{x}, \bar{y}$ as constants or nulls. These give us complete definitions of $Q^+$ and $Q^-$ under OWA and CWA.

**Theorem 4.** *The definitions of $Q^+$ and $Q^-$ above provide efficient sound answers to* FO *queries under* OWA *and* CWA. *The data complexity of such queries is in* $\mathrm{AC}^0$.

**Example** Consider the difference query $Q(\bar{x}) = R(\bar{x}) \wedge \neg S(\bar{x})$ that is among the most troublesome operations for relational query evaluation with nulls [22, 14, 29].

Then the query $Q^+(\bar{x})$ is $R(\bar{x}) \wedge S^-(\bar{x}) \wedge \mathsf{const}(\bar{x})$. Thus, under OWA, $S^-$ and hence $Q^+$ is equivalent to false, which is to be expected, as under OWA the difference query returns the empty set. Under CWA, on the other hand, $Q^+$ computes the set of constant tuples in $R$ which do not match any tuple in $S$.

With $Q^-$, we can also infer useful negative knowledge. Applying the rules, $Q^-(\bar{x}) = R^-(\bar{x}) \vee (S(\bar{x}) \wedge \mathsf{const}(\bar{x}))$. Thus, under OWA it becomes $S(\bar{x}) \wedge \mathsf{const}(\bar{x})$ and we get information that constant tuples in $S$ will never be in the answer, something that traditional certain answers will miss. Under CWA, we also see that tuples not mapped into tuples of $R$ (i.e., $R^-$) can never be query answers.

These are exactly the query results one would expect, and they are obtained by a direct application of transformations giving us queries $Q^+$ and $Q^-$.

## 7 Conclusion

When answering queries over incomplete data, one should concentrate not only on what is guaranteed to be true, but also on what is guaranteed to be false, i.e., negative information. Finding such negative information however is often ignored. We showed how to apply the framework for dealing with incompleteness based on semantics, knowledge, and ordering, to define negative information that can with certainty be inferred about query answers. We showed how to use basic properties of such negative information to find a good representational mechanism for relational query answering, resulting in a natural, but hitherto not widely used mechanism of incomplete tuples. To prove its applicability, we demonstrated an efficient procedure for computing positive and negative knowledge for all FO queries over relational databases.

As next steps, we would like to see how these notions behave in standard applications of incompleteness (integration, inconsistency, etc.), relate them to other approximate query answering notions, both in databases [16, 23, 15, 34] and in AI [27, 32], and to existing approaches that explain why tuples do not appear in query answers [36, 21]. As for quality of approximations of certain answers, these are best confirmed experimentally, as was demonstrated recently [19].

**Acknowledgments** I am grateful to anonymous referees for their comments. This work was partly supported by EPSRC grants J015377 and M025268.

## References

1. S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78(1):158–187, 1991.
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
3. S. Abiteboul, L. Segoufin, and V. Vianu. Representing and querying XML with incomplete information. *ACM TODS*, 31(1):208–254, 2006.
4. M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.

5. P. Barceló, L. Libkin, A. Poggi, and C. Sirangelo. XML with incomplete information. *J. ACM*, 58(1), 2010.
6. P. Barceló, L. Libkin, and J. Reutter. Querying regular graph patterns. *J. ACM*, 61(1), 2014.
7. L. Bertossi. *Database Repairing and Consistent Query Answering*. Morgan&Claypool Publishers, 2011.
8. M. Bienvenu, B. ten Cate, C. Lutz, and F. Wolter. Ontology-based data access: a study through disjunctive datalog, CSP, and MMSNP. *ACM TODS* 39(4) (2014).
9. P. Buneman, A. Jung, A. Ohori, Using powerdomains to generalize relational databases. *Theoretical Computer Science* 91 (1) (1991) 23–55.
10. A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*, pages 260–271, 2003.
11. 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):385–429 (2007).
12. K. Compton. Some useful preservation theorems. *Journal of Symbolic Logic*, 48(2):427–440, 1983.
13. M. Console, P. Guagliardo, L. Libkin. Approximations and refinements of certain answers via many-valued logics. In *KR 2016*, pages 349-358.
14. C. J. Date and H. Darwen. *A Guide to the SQL Standard*. Addison-Wesley, 1996.
15. R. Fink and D. Olteanu. On the optimal approximation of queries using tractable propositional languages. In *ICDT*, pages 174–185, 2011.
16. M. Garofalakis and P. Gibbons. Approximate query processing: taming the terabytes. In *VLDB*, 2001.
17. A. Gheerbrant, L. Libkin, and C. Sirangelo. Naïve evaluation of queries over incomplete databases. *ACM TODS*, 39(4):231, 2014.
18. A. Gheerbrant, L. Libkin. Certain answers over incomplete XML documents: extending tractability boundary. *Theory Comput. Syst.* 57(4): 892-926 (2015).
19. P. Guagliardo, L. Libkin. Making SQL queries correct on incomplete databases: a feasibility study. In *PODS 2016*, pages 211–223.
20. C. Gunter. *Semantics of Programming Languages*. The MIT Press, 1992.
21. M. Herschel and M. Hernández. Explaining missing answers to SPJUA queries. *PVLDB*, 3(1):185–196, 2010.
22. T. Imielinski and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.
23. Y. Ioannidis. Approximations in database systems. In *ICDT*, pages 16–30, 2003.
24. H. Klein. On the use of marked nulls for the evaluation of queries against incomplete relational databases. In *Fundamentals of Information Systems*, T. Polle, T. Ripke, and K. Schewe, Eds. Kluwer, 81–98.
25. R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to ontology-based data access. In *IJCAI*, pages 2656–2661, 2011.
26. M. Lenzerini. Data integration: a theoretical perspective. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002.
27. H. Levesque. A completeness result for reasoning with incomplete first-order knowledge bases. In *KR*, pages 14–23, 1998.
28. L. Libkin. Certain answers as objects and knowledge. *Artif. Intell.* 232 (2016), 1–19.
29. L. Libkin. SQL's three-valued logic and certain answers. *ACM TODS* 41(1) (2016).
30. W. Lipski. On semantic issues connected with incomplete information databases. *ACM TODS*, 4(3):262–296, 1979.
31. W. Lipski. On relational algebra with marked nulls. In *PODS 1984*, pages 201–203.
32. Y. Liu and H. Levesque. A tractability result for reasoning with incomplete first-order knowledge bases. In *IJCAI*, pages 83–88, 2003.

33. R. Reiter. Towards a logical reconstruction of relational database theory. In *On Conceptual Modelling*, pages 191–233, 1982.
34. R. Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *J. ACM*, 33(2):349–370, 1986.
35. R. Rosati. On the decidability and finite controllability of query processing in databases with incomplete information. In *PODS*, pages 356–365, 2006.
36. O. Shmueli and S. Tsur. Logical diagnosis of LDL programs. In *ICLP*, pages 112–129, 1990.