Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)
Main Track

Knowledge-Preserving Certain Answers for SQL-like Queries

Etienne Toussaint[1] , **Paolo Guagliardo**[1] , **Leonid Libkin**[1,2,3]
[1]University of Edinburgh
[2]ENS-Paris, PSL
[3]Neo4j
{etienne.toussaint, paolo.guagliardo, libkin}@ed.ac.uk

Abstract

Answering queries over incomplete data is based on finding answers that are certainly true, independently of how missing values are interpreted. This informal description has given rise to several different mathematical definitions of certainty. To unify them, a framework based on "explanations", or extra information about incomplete data, was recently proposed. It partly succeeded in justifying query answering methods for relational databases under set semantics, but had two major limitations. First, it was firmly tied to the set data model, and a fixed way of comparing incomplete databases with respect to their information content. These assumptions fail for real-life database queries in languages such as SQL that use bag semantics instead. Second, it was restricted to queries that only manipulate data, while in practice most analytical SQL queries invent new values, typically via arithmetic operations and aggregation.

To leverage our understanding of the notion of certainty for queries in SQL-like languages, we consider incomplete databases whose information content may be enriched by additional knowledge. The knowledge order among them is derived from their semantics, rather than being fixed a priori. The resulting framework allows us to capture and justify existing notions of certainty, and extend these concepts to other data models and query languages. As natural applications, we provide for the first time a well-founded definition of certain answers for the relational bag data model and for value-inventing queries on incomplete databases, addressing the key shortcomings of previous approaches.

1 Introduction

Many data analysis tasks that occur in traditional AI applications rely on the standard trusted database technology to produce final solutions to a problem. Most of such applications in a very natural way produce data that is incomplete; in particular, data with missing values represented as `NULL`s in SQL. Even though SQL has no explicit support for dealing with incompleteness and does not even specify whether `NULL` denotes a non-applicable or missing value, one can always consider a setting where non-applicable nulls have been removed – e.g., by using the techniques of (Franconi and Tessaris 2012) – and `NULL`s only denote missing values. These commonly occur in ontology-mediated query answering (Bienvenu and Ortiz 2015; Poggi et al. 2008; Kontchakov et al. 2011), data integration (Lenzerini 2002; Calì, Lembo, and Rosati 2003), data exchange (Arenas et al. 2014), and handling inconsistent data (Bertossi 2011).

Reliance on database techniques imposes significant limits on queries that can be answered correctly. Indeed, an experimental study showed that common benchmark queries with negation produce unreliable results (Guagliardo and Libkin 2016). Unfortunately, our technical arsenal for correct query answering on incomplete databases is still rather limited. This is especially so when it comes to queries most commonly found in real-life scenarios.

Since the early days of database field, the standard approach to answering queries over incomplete databases has been based on finding answers one can be *certain* about, regardless of the interpretation of the missing data. This notion was introduced about 40 years ago (Grant 1977; Jr. 1979; Codd 1979) and has since been studied on its own, leading to varying mathematical definitions of certainty (Grahne 1991; van der Meyden 1998; Jr. 1984; Imielinski and Jr. 1984; Abiteboul, Segoufin, and Vianu 2006; Libkin 2016a; Libkin 2016b). All these notions are traditionally tested against a single setting: first-order (or closely related) queries over relational databases, interpreted under set semantics. While understanding certainty and its computational properties in this setting was very useful (Console et al. 2020), it nonetheless falls short of what one needs to deal with in realistic everyday queries, like those written in SQL.

The key shortcomings of existing techniques are of two kinds. To start with, real-life databases operate with *bags* rather than sets. Syntactically straightforward extensions of certainty notions have been studied over bags (Console, Guagliardo, and Libkin 2017) but they were not properly justified unlike their set-theoretic counterparts (and we shall see later that this indeed leads to serious problems with existing definitions). Furthermore, existing notions only work for queries that essentially manipulate data. Real-life queries also *generate* new data values, by means of, for example, arithmetic operations or aggregates. In fact, in the standard TPC-H benchmark for evaluating SQL-database performance, over 90% of queries are value-inventing (TPC 2014), and all of them use bag semantics.

Our goal is thus twofold. We want to build an abstract framework for justifying the notion of certainty, and upon validating it, we shall use it to explain what certainty is for

realistic SQL-like queries that use bag semantics and value-invention. By validating we mean that it should capture existing notions in the setting where they are well understood, namely relational databases under set semantics, and no value invention in queries.

We follow an approach advocated in (Libkin 2016a) that has the advantage of being applicable in different data models. The key concept is that of informativeness of databases: a database x is more informative than a database y if all the possible worlds it represents are also possible worlds represented by y. Intuitively the more informative a database is, the fewer worlds it may represent and the less ambiguous it is. Then the certain answer to a query on an incomplete database x is the most informative database which is not more informative than the query answer on every possible world of x. This however may be too permissive as it misses the reason, or *explanation*, why a complete database is a possible world for an incomplete database x. In general an explanation can turn an incomplete object into a more informative object; explanations compose, and certain answers must preserve the informativeness order provided by the explanation. Such an order essentially says that applying an explanation to a database results in a more informative database. Using this idea, (Amendola and Libkin 2018) developed a framework that was capable of explaining the commonly used definition of certainty of (Jr. 1984), for relational database queries under set semantics and informativeness orders naturally imposed by such a model. The main drawback of the approach is that it is too closely tied to a particular data model (relations as sets) and the informativeness orders it imposes, and to particular features of query languages (no bags, no value invention).

To overcome those problems, we refine the information-based framework in a way that opens up an approach to defining certainty for most typical queries that occur in SQL-like languages over real-life databases. The key concept is that of *knowledge preservation* to improve the information content of answers to queries on incomplete databases. Intuitively, the interpretation of missing data may be specified by some additional knowledge, and hence the information content of a database is defined with respect to this knowledge. Consequently, certainty of an answer to a query q on a database x means two things. First, it is no more informative than answers to q in all possible worlds x. Second, if c is a possible world for x obtained by providing some new knowledge, then the answer to q on c must also be a possible world for the answer to q on x with this new knowledge.

The main distinguishing feature of this approach is that we derive the knowledge-preserving information pre-order from the semantics of data, rather than have it as a basic notion in the model. The latter restricted us to a handful of cases where such an ordering already existed, such as familiar open- and closed-world interpretation of incomplete databases under set semantics. The new approach allows us to unify already existing notions of certain answers without using any specific database model. The framework is very natural to instantiate for other data models and for expressive query languages. We demonstrate it by showing that it allows us for the first time to give a well-founded notion of certain answers on bag relational databases. We demonstrate its additional power with respect to query language features by showing that we can define, again for the first time, a notion of certain answers for value-inventing queries. We can further use the approach to devise an implementation-friendly approximation scheme for such certain answers.

Organization Preliminary definitions and notation are introduced in Section 2, and in Section 3 we present our framework based a notion of certain answers that preserve knowledge. We then apply the general framework to two cases of practical interest: bag relational databases (Section 4) and value-inventing queries (Section 5). We conclude with final remarks and a discussion of future work in Section 6.

2 Preliminaries

We begin by introducing the basic notions and notation that we will use throughout the paper. The abstract framework is largely adapted from (Libkin 2016a), which itself was based on earlier work utilizing techniques from programming semantics and domain theory in handling incompleteness in databases such as (Buneman, Jung, and Ohori 1991; Rounds 1991; Libkin 1995). We refer the reader to (Libkin 2011; Libkin 2016a) for more detailed information about such connections.

2.1 Information-Based Certain Answers

A *database domain* $\mathbf{D}$ is a triple $(\mathbb{I}, \mathbb{C}, [\![\cdot]\!])$, where $\mathbb{I}$ is a set of database objects,[1] $\mathbb{C} \subseteq \mathbb{I}$ is the set of complete databases in $\mathbb{I}$ and $[\![\cdot]\!] : \mathbb{I} \to \mathcal{P}(\mathbb{C})$ is the semantic function that assigns a set of complete databases (possible worlds) to a database object. For notational convenience, we also use $\mathbb{I}_{\mathbf{D}}$, $\mathbb{C}_{\mathbf{D}}$, $[\![\cdot]\!]_{\mathbf{D}}$ to refer to the set of incomplete databases, the set of complete databases, and the semantic function, respectively, of a database domain $\mathbf{D}$. Intuitively, the more possible worlds an incomplete database represents, the more ambiguous it is. To make this intuition formal, with a database domain $\mathbf{D} = (\mathbb{I}, \mathbb{C}, [\![\cdot]\!])$ we associate an information pre-order $\preceq_{\mathbf{D}}$ on $\mathbb{I}$, defined as follows: $x \preceq_{\mathbf{D}} y$ iff $[\![y]\!] \subseteq [\![x]\!]$. That is, a database $x \in \mathbb{I}$ is *less informative* than a database $y \in \mathbb{I}$ if every possible world of y is also a possible world of x. This is a pre-order (i.e., a reflexive transitive relation). The associated equivalence relations, when $x \preceq_{\mathbf{D}} y$ and $y \preceq_{\mathbf{D}} x$, is denoted by $x \equiv_{\mathbf{D}} y$. We shall be using the notation $[\![X]\!]$ for $\bigcup_{x \in X} [\![x]\!]$.

Given two database domains $\mathbf{S}$ (for source) and $\mathbf{T}$ (for target), a *query* from $\mathbf{S}$ to $\mathbf{T}$ is a mapping q from $\mathbb{C}_{\mathbf{S}}$ to $\mathbb{C}_{\mathbf{T}}$, i.e., it maps complete databases of a source database domain $\mathbf{S}$ to complete databases of a target database domain $\mathbf{T}$. The standard way to define query answers on *incomplete* databases is by considering answers that are true in all possible worlds, that is, answers that are no more informative than the answers to the query on all possible worlds represented by the incomplete database. To this end, if C is a set

[1] The term "database objects" here refers to generic structures, such as relational or graph databases, that store data. Some of these objects are assumed to provide complete information, while others do not.

of complete databases, we let

$$q(C) = \{q(c) \mid c \in C\}$$

Then, the *information-based certain answer* to a query q from $\mathbf{S}$ to $\mathbf{T}$ on an incomplete source database $x \in \mathbb{I}_{\mathbf{S}}$ is the greatest lower bound (denoted by glb) of the query answers on the possible worlds of x with respect to the information pre-order:

$$\mathsf{cert}_{\Box}(q, x) = \mathsf{glb}_{\preceq_{\mathbf{T}}}\, q([\![x]\!]_{\mathbf{S}}) \tag{1}$$

In other words, the information based-certain answer is the most informative database that is consistent, w.r.t. the information pre-order, with every possible answer to the query. In particular, we have $q([\![x]\!]_{\mathbf{S}}) \subseteq [\![\mathsf{cert}_{\Box}(q, x)]\!]_{\mathbf{T}}$. Note that for general pre-orders, the greatest lower bound may not exist, especially for infinite sets, and even when it exists in (1), it is defined up to the $\equiv_{\mathbf{T}}$ equivalence relation.

We will see an example of information-based certain answers after introducing relational database domains.

2.2 Relational Databases

In the context of relational databases, the standard model for representing incomplete data is that of *marked nulls*, where databases are populated by two kinds of values, *constants* and *nulls*. These come from two disjoint countably infinite sets, Const and Null, respectively. We denote the elements of Null using the symbol $\perp$ with subscripts.

A relational database schema is a finite set of table names with associated arities, and a k-ary table is a finite bag (a.k.a. multiset) of k-tuples over $\mathsf{Const} \cup \mathsf{Null}$. The number of occurrences of a tuple $\bar{t}$ in a table T is denoted by $\#(R, \bar{t})$. Then, a relational database d over a given schema maps each table name R in the schema to a table R^d of appropriate arity. We write $\mathsf{Const}(d)$ and $\mathsf{Null}(d)$ for the set of constants and nulls occurring in d, respectively. For convenience of notation, we sometimes represent a relational database as a bag of facts; e.g., $d = \{\!\{R(1, \perp_1), S(\perp_1, 2), R(1, \perp_1)\}\!\}$ is the database d such that $R^d = \{\!\{(1, \perp_1), (1, \perp_1)\}\!\}$ and $S^d = \{\!\{(\perp_1, 2))\}\!\}$.

A *valuation* is a partial function v from Null to Const. If v is defined on all elements of $\mathsf{Null}(d)$, we say that v is d-*complete*. For $\bar{t} = (t_1, \ldots, t_k)$, we let $v(\bar{t})$ denote the tuple $\big(\tilde{v}(t_1), \ldots, \tilde{v}(t_k)\big)$, where $\tilde{v}$ is the total function on $\mathsf{Null} \cup \mathsf{Const}$ such that $\tilde{v}(t) = v(t)$ whenever $v(t)$ is defined, and $\tilde{v}(t) = t$ otherwise.

When tables are constrained to be sets, the result of applying a valuation v to a table T is the table $v(T) = \{v(\bar{t}) \mid \bar{t} \in T\}$, and this extends to databases in the natural way. When tables are bags, things are slightly more complicated, as we shall see in Section 4.

A database domain $(\mathbb{I}, \mathbb{C}, [\![\cdot]\!])$ is *relational* if $\mathbb{I}$ is a set of relational databases and, for every $d \in \mathbb{C} \subseteq \mathbb{I}$, $\mathsf{Null}(d) = \varnothing$. When tables are constrained to be sets, the two most common semantics of incompleteness, referred to as the semantics under open-world and closed-world assumptions (OWA and CWA), are defined as follows (cf. (Imielinski and Jr. 1984; Grahne 1991)):

$$[\![d]\!]^{\text{OWA}} = \{d' \text{ complete} \mid \exists \text{ valuation } v \text{ s.t. } v(d) \subseteq d'\}$$
$$[\![d]\!]^{\text{CWA}} = \{d' \text{ complete} \mid \exists \text{ valuation } v \text{ s.t. } v(d) = d'\}$$

In other words, under CWA we simply replace nulls with constants, while under OWA we can also add extra tuples.

On relational databases, a query answer can be improved in two ways: by finding more tuples, or by instantiating nulls with values. This corresponds to the open-world semantics of incompleteness. Therefore, we assume that the the target domain of queries always uses the OWA semantics, while the source can use either the CWA or OWA semantics; see (Console et al. 2020).

Below, we recall two commonly used notions of certain answers due to (Jr. 1984; Libkin 2016b; Console, Guagliardo, and Libkin 2017). These apply for queries from relational database domains under OWA or CWA to relational database domains under OWA.

The *intersection-based certain answer* to a query q on a relational database d is the intersection of the answers to q on every possible world represented by d:

$$\mathsf{cert}_{\cap}(q, d) = \bigcap \{q(d') \mid d' \in [\![d]\!]\} \tag{2}$$

The *certain answer with nulls* to a query q on a relational database d is the table $\mathsf{cert}_{\perp}(q, d)$ such that, for every d-complete valuation v and for every $c \in [\![v(d)]\!]$, the multiplicity of each tuple in $v\big(\mathsf{cert}_{\perp}(q, d)\big)$ is less than or equal to its multiplicity in $q(c)$:

$$\#\big(\mathsf{cert}_{\perp}(q, d), \bar{t}\,\big) = \min\big\{\, \#\big(q(c), v(\bar{t})\big) \mid c \in [\![v(d)]\!], \\ v \text{ is a } d\text{-complete valuation}\big\} \tag{3}$$

We remark that the above definition is implicit in (Console, Guagliardo, and Libkin 2017) and it is a natural extension of certain answers with nulls for sets (Libkin 2016b) to bags.

The information-based notion of certain answers (1) is able to capture intersection-based certain answers. Let $\mathbf{S}$ be a relational database domains and $\mathbf{T}$ a complete relational database domain, i.e., relational databases without null values. Then, for every query q from $\mathbf{S}$ to $\mathbf{T}$, either both the information-based certain answer $\mathsf{cert}_{\Box}(q, d)$ and the intersection-based certain answer $\mathsf{cert}_{\cap}(q, d)$ exist and coincide, or neither of them exists.

On the other hand, the information-based notion of certain answers fails to capture certain answers with nulls. This is due to the fact that it does not keep track of what nulls are mapped to, as the following example shows.

Example 1. Consider the database $d = \{R(\perp_1, 1)\}$ and a query q that returns the first and second columns of R plus an extra column with their sum. The most relevant answer one could expect is $(\perp_1, 1, \perp_1 + 1)$. But $\mathsf{cert}_{\perp}(q, d)$ is the empty table, because it cannot capture the value $\perp_1 + 1$. Indeed, for every null $\perp$ there exits a valuation v such that $v\big((\perp_1, 1, \perp)\big)$ is not in $q\big(v(d)\big)$. On the other hand, the information-based notion is more permissive, as $\mathsf{cert}_{\Box}(q, d)$ gives us a tuple of the form $(\perp_i, 1, \perp_j)$, where $\perp_i$ and $\perp_j$ are fresh nulls. However, since null labels are not preserved, there is no relationship between $\perp_i$ and $\perp_j$ in the output, and $\perp_1$ in the input. In particular, it is not enforced that the value $\perp_i$ in the output and the value $\perp_1$ in the input must be equal (and that $\perp_j$ is equal to $\perp_1 + 1$).

We will later build on the above example to also show that neither information-based certain answers nor certain answers with nulls are entirely satisfactory. To reconcile the two notions, we introduce the concept of knowledge preservation.

3 Knowledge-preserving Certain Answers

As explained earlier, an incomplete database x is less informative than y if every possible world of y is also a possible world of x. Suppose now that we discover some new information that reduces the ambiguity of x by eliminating some possible worlds; nothing ensures that this knowledge would also reduce the incompleteness of y. Indeed, it may well be the case that, in the presence of some additional knowledge, x would become more informative than y.

To make this intuition formal, we define the notion of *universal knowledge* $\mathcal{K}$. Its elements are viewed as pieces of knowledge that can be applied to a database to make it potentially more informative. Such pieces of knowledge can be concatenated: applying $\omega\omega'$ means applying ω first and then applying ω' to the result of applying ω. Finally we have the empty knowledge ε: applying it to any database x does not change x. That is, $\mathcal{K}$ is a *monoid* as it has a binary concatenation operation $\omega\omega'$ (which we assume to be associative) and the identity ε satisfying $\varepsilon\omega = \omega\varepsilon = \omega$.

Then, for a database domain $(\mathbb{I}, \mathbb{C}, [\![\cdot]\!])$, the $\mathcal{K}$*-semantics* of an incomplete database $x \in \mathbb{I}$ under knowledge $\omega \in \mathcal{K}$ is the set $[\![x]\!]^\omega$ of complete databases in $\mathbb{C}$ represented by x when ω is known. We assume that, for any given database domain, such a semantic function from $\mathbb{I} \times \mathcal{K}$ to $2^{\mathbb{C}}$ always exists and it is such that $[\![x]\!]^\varepsilon = [\![x]\!]$ for every $x \in \mathbb{I}$. Intuitively, the empty knowledge does not give us any extra information about the semantics of incomplete databases.

As $\mathcal{K}$ contains all possible knowledge, it would be reasonable to assume that the $\mathcal{K}$-semantics, while always existing, is not always fully known. Thus, we consider sub-monoids of $\mathcal{K}$, in particular those whose knowledge increases the information content of database objects.

Definition 1. *A sub-monoid $\mathcal{A}$ of the universal knowledge $\mathcal{K}$ is* additional knowledge *for a database domain $\mathbf{D}$ if*

$$\varnothing \subsetneq [\![x]\!]_{\mathbf{D}}^{\omega\omega'} \subseteq [\![x]\!]_{\mathbf{D}}^{\omega} \tag{4}$$

for every $x \in \mathbb{I}_{\mathbf{D}}$ and for every $\omega, \omega' \in \mathcal{A}$.

Intuitively, the information content of an incomplete database can only increase with the additional knowledge in $\mathcal{A}$ and is consistent with it. Note that $\{\varepsilon\}$ is additional knowledge for every database domain and corresponds to the situation in which we only have "empty" knowledge.

We can now define the relative informativeness of incomplete databases in the presence of additional knowledge.

Definition 2. *Let $\mathcal{A}$ be additional knowledge for $\mathbf{D}$, and let $x, y \in \mathbb{I}_{\mathbf{D}}$. We say that x is* less $\mathcal{A}$-informative than *y, and write $x \preceq_{\mathbf{D}}^{\mathcal{A}} y$, if $[\![y]\!]_{\mathbf{D}}^{\omega} \subseteq [\![x]\!]_{\mathbf{D}}^{\omega}$ for every $\omega \in \mathcal{A}$.*

In other words, x is less informative than y under the additional knowledge $\mathcal{A}$ if, for every piece of knowledge $\omega \in \mathcal{A}$, each possible world of y under ω is also a possible world of x under ω. We omit the subscript in $\preceq_{\mathbf{D}}^{\mathcal{A}}$, and simply write $\preceq^{\mathcal{A}}$, when the database domain is clear from the context.

As shown below, additional knowledge conservatively extends the information-based framework of (Libkin 2016a): if we do not have any knowledge beyond ε, we get the standard information pre-order $\preceq$; but the more we discover – by means of additional knowledge – of the $\mathcal{K}$-semantics of incomplete databases, the less permissive the information preorder becomes.

Proposition 1. *The following are true:*

(a) $\preceq^{\mathcal{A}}$ is a pre-order;
(b) $\preceq^{\{\varepsilon\}}$ is a equivalent to $\preceq$;
(c) If $x \preceq^{\mathcal{A}} y$, then $x \preceq^{\mathcal{B}} y$ for every sub-monoid $\mathcal{B}$ of $\mathcal{A}$.

With this in place, our goal is to capture answers that are consistent with the answers on every possible world under every extra knowledge.

Definition 3 (Knowledge-preserving certain answer)**.** *Let q be a query from $\mathbf{S}$ to $\mathbf{T}$, let $x \in \mathbb{I}_{\mathbf{S}}$, and let $\mathcal{A}$ be additional knowledge for both $\mathbf{S}$ and $\mathbf{T}$. The* $\mathcal{A}$-preserving certain answer *to q on x, denoted by $\mathsf{cert}_{\mathcal{A}}(q, x)$, is the most informative database with respect to $\preceq^{\mathcal{A}}$ that satisfies the following:*

$$q\big([\![x]\!]_{\mathbf{S}}^{\omega}\big) \subseteq [\![\mathsf{cert}_{\mathcal{A}}(q, x)]\!]_{\mathbf{T}}^{\omega} \tag{5}$$

for every $\omega \in \mathcal{A}$.

This notion generalizes the information-based certain answer $\mathsf{cert}_{\square}$ and it relies only on the existence of a possible-worlds semantics for the additional knowledge in $\mathcal{A}$.

The $\mathcal{A}$-preserving certain answer can also be interpreted as a kind of "synchronized" greatest lower bound, with respect to $\preceq^{\mathcal{A}}$, of the query answers on the possible worlds of the input:

$$\mathsf{cert}_{\mathcal{A}}(q, d) = \mathsf{glb}_{\preceq^{\mathcal{A}}}\, q(x)$$

where, for every knowledge $\omega \in \mathcal{K}$, the $\mathcal{K}$-semantics of the artifact $q(x)$ is given as $[\![q(x)]\!]_{\mathbf{T}}^{\omega} = q([\![x]\!]_{\mathbf{S}}^{\omega})$.

The expected behavior of query answering under incomplete information is that more informative query inputs yield more informative query outputs. We show that this is indeed the case for certain answers that preserve additional knowledge. Moreover, the information-based certain answers are precisely the certain answers that preserve the empty knowledge.

Proposition 2 (Information preservation)**.** *Let $\mathcal{A}$ be a sub-monoid of $\mathcal{K}$, and let q be a query from $\mathbf{S}$ to $\mathbf{T}$. Then, for all $x, y \in \mathbb{I}_{\mathbf{S}}$ and every $\omega \in \mathcal{A}$, all of the following hold:*

(a) If $\mathsf{cert}_{\mathcal{A}}(q, x)$ and $\mathsf{cert}_{\mathcal{A}}(q, y)$ exist and $x \preceq^{\mathcal{A}} y$, then $\mathsf{cert}_{\mathcal{A}}(q, x) \preceq^{\mathcal{A}} \mathsf{cert}_{\mathcal{A}}(q, y)$;
(b) $\mathsf{cert}_{\mathcal{A}}(q, x) \preceq^{\mathcal{B}} \mathsf{cert}_{\mathcal{B}}(q, x)$ for every $\mathcal{B} \subseteq \mathcal{A}$;
(c) $\mathsf{cert}_{\square}(q, x) = \mathsf{cert}_{\{\varepsilon\}}(q, x)$.

The above also implies that information-based certain answers are the most informative answers one could expect with empty knowledge; however, they do not preserve non-empty knowledge in general. The following example shows why this is important.

Example 2. Consider again the setting of Example 1, and the extra piece of knowledge ω encoding the fact that the value of $\bot_j$ is 0. Clearly, the information content of the original database d is not increased by this new knowledge, since $\bot_j \notin \mathsf{Null}(d)$. But $\bot_j$ occurs in the information-based certain answer $(\bot_i, 1, \bot_j)$, whose possible worlds under ω always contain the tuple $(\bot_i, 1, 0)$. Thus, the information-based certain answer under additional knowledge is not less informative than all possible answers under the same knowledge, and it may contain false tuples.

What this means is that information-based certain answers become irrelevant once we discover additional knowledge: they must be recomputed on a more informative database, if it exists. Knowledge-preserving certain answers do not have this shortcoming, as we show next.

Theorem 1. *Let q be a query from $\mathbf{S}$ to $\mathbf{T}$, let $x \in \mathbb{I}_{\mathbf{S}}$, and let $\mathcal{A}$ be additional knowledge for both $\mathbf{S}$ and $\mathbf{T}$. If $\mathsf{cert}_{\mathcal{A}}(q, x)$ exists, then*

$$[\![\mathsf{cert}_{\mathcal{A}}(q, x)]\!]^{\omega}_{\mathbf{T}} \subseteq q([\![x]\!]^{\omega}_{\mathbf{S}}) \tag{6}$$

for every $\omega \in \mathcal{A}$. Moreover, there exist a query q, database $x \in \mathbb{I}_{\mathbf{S}}$ and $w \in \mathcal{A}$ such that:

(a) $[\![\mathsf{cert}_{\square}(q, x)]\!]^{\omega}_{\mathbf{T}} \not\subseteq q([\![x]\!]^{\omega}_{\mathbf{S}})$, *and*

(b) $[\![\mathsf{cert}_{\mathcal{A}}(q, x)]\!]^{\omega}_{\mathbf{T}} \subseteq [\![\mathsf{cert}_{\square}(q, x)]\!]_{\mathbf{T}}$.

Informally, knowledge-preserving certain answers remain relevant and consistent when new information is discovered. Moreover, there are queries for which this notion gives more informative answers under additional knowledge.

Next, we apply our general framework to relational databases, under the bag semantics that is the standard in all relational products. We do so in a setting where additional knowledge is given by partial valuations of nulls. We then show that the well-known certain answers with nulls corresponds to the valuation-preserving certain answers. We also use our results to justify notions used hitherto in the set semantics context.

4 Certainty in Bag Relational Databases

Recall that an incomplete relational database is a finite set of tables populated by constants and nulls, and its semantics of incompleteness is given by instantiating nulls with constants by means of valuations.

Definition 4. *The monoid of partial valuations $\mathcal{V} \subseteq \mathcal{K}$ consists of all partial functions from* Null *to* Const. *Moreover, for every $v, v' \in \mathcal{V}$, the $\mathcal{K}$-semantics of incomplete relational databases is $[\![d]\!]^{vv'} = [\![v'(v(d))]\!]$.*

We immediately have that the knowledge encoded by $\mathcal{V}$ is additional to the relational database domain for both closed and open world semantics. However, so far we have not explained how a valuation v is applied to a relational database d. When tables T are constrained to be sets, there is no ambiguity on how to construct $v(T)$. But when they are bags, there are several possibilities, each leading to different semantics (Hernich and Kolaitis 2017).

4.1 Collapsing and Additive Semantics

The first construction we study prescribes that, when a valuation is applied to a table, distinct tuples that become equal (i.e., unify) under the valuation are "collapsed" together, so that the maximum number of occurrences of each such tuple appears in the result.

Definition 5. *The* collapsing application *of a valuation v on a table T is the table $v(T)$ such that for every tuple $\bar{t}$*

$$\#\big(v(T), \bar{t}\,\big) = \max\big\{\, \#(T, \bar{u}) \mid v(\bar{u}) = \bar{t} \big\}$$

For example, consider a table T given by the following bag:

$$\{\!\{(1, 2), (\bot_1, 2), (1, \bot_2), (1, \bot_2)\}\!\} \tag{7}$$

and a valuation v such that $v(\bot_i) = i$ for every i. Then, the collapsing application of v to T is the table $v(T)$ given by the bag $\{\!\{(1, 2), (1, 2)\}\!\}$.

Intuitively, when valuations are applied in such a way, incomplete facts only represent new information if they do not unify with already existing data. Below we show that, under this semantics of valuations, the certain answers with nulls coincide with the valuation-preserving ones.

Proposition 3. *Let $\mathbf{S}$ and $\mathbf{T}$ be relational database domains under the collapsing application of valuations. Then, for every query q from $\mathbf{S}$ to $\mathbf{T}$ and for every $d \in \mathbb{I}_{\mathbf{S}}$, either both $\mathsf{cert}_{\mathcal{V}}(q, d)$ and $\mathsf{cert}_{\bot}(q, d)$ exist and coincide, or neither of them exists.*

We remark that the above result holds for databases under the bag data model, but also – in particular – when tables are constrained to be sets.

Since the collapsing semantics negates the importance of a tuple's identity by disregarding – to some extent – its multiplicity, the most commonly used semantics is instead one where the multiplicities of tuples that become equal under a valuation are added up in the result.

Definition 6. *The* additive application *of a valuation v on a table T is the table $v(T)$ such that for every tuple $\bar{t}$*

$$\#\big(v(T), \bar{t}\,\big) = \sum_{\bar{u}\colon\, v(\bar{u}) = \bar{t}} \#(T, \bar{u})$$

To see the difference with the collapsing semantics, consider again the table T given by the bag in (7), and the same valuation v such that $\bot_i \mapsto i$. The additive application of v to T results in a table $v(T)$ consisting of 4 (as opposed to 2) occurrences of the tuple $(1, 2)$. When valuations are applied in such an additive way, incomplete facts always represent new information even if they unify with already existing data, in contrast with the collapsing semantics discussed earlier.

However, while more natural, this semantics of valuations leads to problems in terms of certain answers, as shown below.

Theorem 2. *There exists a union of conjunctive queries with negation where the source and target database domain are relational database with additive semantics, such that:*

(a) *the valuation-preserving certain answer does not exist;*

(b) *the $\{\varepsilon\}$-preserving certain answer does not exist.*

(c) *the certain answer with nulls is not less informative than every possible answer.*

Below we give an example in which the certain answer with nulls to a query q on a database d is not less informative than the answer to q on some possible world of d.

Example 3. Consider the following database D:

R

A	B	C
1	$\perp_1$	2
$\perp_2$	2	1

S

A	B	C	D
1	2	$\perp_2$	$\perp_1$

T

A	B
1	2

and the following relational algebra query q:

$$\pi_{A,B}\big(\sigma_{A\neq C\wedge B\neq C}(R)\big) \uplus \pi_{A,B}\big(\sigma_{A=C\wedge B=D}(S)\big) \uplus T$$

Then, $\mathsf{cert}_\perp(q, D)$ is the bag $\{\!\{(1,2),(1,\perp_1),(\perp_2,2)\}\!\}$. If we take a valuation v such that $v(\perp_1) = 2$ and $v(\perp_2) = 1$, then $\{\!\{(1,2);(1,2)\}\!\}$ belongs to $q\big([\![v(D)]\!]\big)$. But with the additive semantics $[\![\mathsf{cert}_\perp(q, D)]\!]$ contains at least three tuples and so it cannot be less informative than $q\big(v(D)\big)$.

The answers $A_1 = \{\!\{(1,2)\}\!\}$ and $A_2 = \{\!\{(1,\perp_i),(\perp_k,2)\}\!\}$ are both less informative than every element of $q([\![d]\!])$. Thus, $\mathsf{cert}_\square(q, D)$ should be more informative than both A_1 and A_2. We clearly have that $A_1 \not\preceq A_2$ and $A_2 \not\preceq A_1$; moreover, it is easy to check that we cannot increase the information content of A_1 or A_2 and still be less informative than the answer to q on every possible world described by D. Therefore, the information-based certain answer does not exist.

The above result tells us that, under additive valuations, the notion of certain answers with nulls is not well-founded, as even for simple queries they may not be less informative than the query answers on every possible world. Thus, some fact in the certain answers with nulls might be false.

Moreover, Example 2 showed that the information-based certain answer can be strictly less informative than the certain answer with nulls, and Proposition 2 states that, for every knowledge, the knowledge-preserving certain answer is no more informative than the information-based certain answer. Therefore, there is no additional knowledge such that knowledge-preserving certain answers capture certain answers with nulls. We argue that this is due to the additive semantics itself.

To see this, consider two incomplete relational databases d_1 and d_2 of the same schema. Then, under OWA and collapsing valuations, we can always build a database d_3 such that $[\![d_3]\!] = [\![d_1]\!] \cap [\![d_2]\!] = [\![d_1 \cup d_2]\!]$, where $\cup$ denotes the union-max operator: $\#(T_1 \cup T_2, \bar{t}) = \max\big(\#(T_1, \bar{t}), \#(T_2, \bar{t})\big)$. This ensures that, if we have a finite number of possible answers to a query, then the certain answers always exist. However, under additive valuations, we lose this property: as shown in Example 3, there exist databases d_1 and d_2 such that, for every database d_3, we have $[\![d_3]\!] \neq [\![d_1]\!] \cap [\![d_2]\!]$. This can be interpreted as a mismatch between the semantics and the set of incomplete objects.: there are not enough objects to capture the complexity of the semantics.

Now that we have identified the problem, we can finally capture the notion of certain answers with nulls by changing the semantics of target. To do this under additive valuations, we need to interpret query answers under the collapsing semantics.

Proposition 4. *Let $\mathbf{S}$ and $\mathbf{T}$ be relational database domains under the additive and collapsing application of valuations, respectively. Then, for every query q from $\mathbf{S}$ to $\mathbf{T}$ and for every database $d \in \mathbb{I}_\mathbf{S}$, either both $\mathsf{cert}_\mathcal{V}(q, d)$ and $\mathsf{cert}_\perp(q, d)$ exist and coincide, or neither of them exists.*

Corollary 1. *The result of Proposition 4 holds, in particular, for relational databases under set, rather than bag, semantics. In that context, the certain answer with nulls becomes:*

$$\mathsf{cert}_\perp(q, d) = \big\{\bar{t} \mid v(\bar{t}) \in q\big(v(d)\big) \text{ for every } d\text{-complete valuation } v\big\}$$

which corresponds to the definition given by (Jr. 1984).

We have been able to justify the notion of certain answers with nulls for both the additive and the collapsing semantics of valuations, but neither is entirely satisfactory. On the one hand, collapsing valuations decreases the expressiveness of the relational data model based on bags. On the other hand, certain answers with nulls under additive valuations must be interpreted using the collapsing semantics, which is counter-intuitive.

We will now propose a new semantics of incompleteness for relational databases that overcomes these shortcomings.

4.2 Mixed Semantics

The main idea behind our proposed semantics is that applying a valuation to a table does not produce *one* table, as is the case with the collapsing and additive semantics, but rather a *set* of tables.

Definition 7. *The* mixed application *of a valuation v to a k-ary table T is the set $v(T)$ consisting of all tables B of arity k such that, for every k-ary tuple $\bar{t}$ of constants and nulls:*

$$\max\big\{\#(B,\bar{u}) \mid v(\bar{u}) = \bar{t}\,\big\} \leq \#(B,\bar{t}) \leq \sum_{\bar{u}:\ v(\bar{u})=\bar{t}} \#(B,\bar{u})$$

This extends to databases d as follows: $v(d)$ is the set of all databases d_1 of the same schema as d such that R^{d_1} belongs to $v(R^d)$ for every table name R.

Finally, the mixed semantics of incompleteness under OWA and CWA are given as follows:

$$[\![d]\!]^{\text{CWA}} = \{d_1 \text{ complete} \mid d_1 \in v(d),\ v \text{ is a valuation}\}$$
$$[\![d]\!]^{\text{OWA}} = \{d_2 \text{ complete} \mid d_2 \supseteq d_1 \in v(d),\ v \text{ is a valuation}\}$$

As the name suggests, the mixed semantics combines the collapsing and the additive ones, by taking into account that incomplete facts unifying with already existing data may or may not represent new information.

Below we show that, when using the mixed semantics of valuations on both the source and target domain of queries, the valuation-preserving certain answers and the certain answers with nulls coincide.

Theorem 3. *Let* $\mathbf{S}$ *and* $\mathbf{T}$ *be relational database domains under the mixed application of valuations. Then, for every query* q *from* $\mathbf{S}$ *to* $\mathbf{T}$ *and for every database* $d \in \mathbb{I}_{\mathbf{S}}$*, either both* $\mathsf{cert}_{\mathcal{V}}(q,d)$ *and* $\mathsf{cert}_{\perp}(q,d)$ *exist and coincide, or neither of them exists.*

Note that when we consider the open world assumption, the collapsing and mixed semantics are equivalent. And for every valuation from $\mathcal{V}$ their $\mathcal{K}$-semantics are also the same. However, as illustrated in Example 4, there exists a query such that valuation-preserving certain answers upon additive, collapsing and mixed semantics are different for closed-world semantics.

Example 4. Let d be a relational database such that

$$R^d = \{\!\{1, \perp_1, \perp_2\}\!\}\,; \quad S^d = \{\!\{1, 1\}\!\}\,;$$

and consider the following relational algebra query q:

$$\pi_{\varnothing}\big((R - S) \uplus (S - R)\big)$$

It is easy to see that the certain answers of q are empty iff there exists a possible world c of d such that $R^c = S^c$. Since S^d does not contain nulls, we focus on R.

Under the additive semantics of valuations, for every possible world c of d there are at least 3 tuples in R^c. As there are always exactly 2 tuples in S^c, we have that $R^c - S^c$ cannot be empty. Thus, the valuation-preserving certain answer to q on d is non-empty.

Under the collapsing semantics of valuations, $\{\!\{1, 1\}\!\} \neq v(R^d)$ for every valuation v because, even if $v(\perp_1) = 1$ or $v(\perp_2) = 1$, they collapse into a single occurrence of value 1. Hence, for every possible world c of d, we have that $S^c - R^c$ cannot be empty, and therefore the valuation-preserving certain answer to q on d is non-empty.

Under the mixed semantics of valuations, we consider every possible multiplicity for each tuple, and $\{\!\{1, 1\}\!\} \in v(R^d)$ when $v(\perp_1) = v(\perp_2) = 1$. Thus, the empty bag is a possible answer to q, and the valuation-preserving certain answer to q on d is empty.

In light of the above, we argue that the semantics of valuations one should use in the context of relational databases is the mixed semantics, because it behaves consistently independently of whether a (relational) database domain is the source or the target of queries. However, under mixed semantics, the valuation-preserving certain answers coincide with the certain answers with nulls, and these are still unsatisfactory as they cannot produce answers involving values that were not present in the original data, as Example 1 illustrates.

We would like to capture all the information-based certain answers, while keeping the valuation-preservation property of the certain answers with nulls. By Proposition 2, we know that with the relational database domain as target it is impossible to be valuation-preserving and more informative at the same time. Therefore, we must modify the target domain; this is what we do in the next section.

5 Certainty for Value-Inventing Queries

To handle queries whose answers may contain values not appearing in the original input, we extend relational databases with the notion of *persistent nulls*. Informally, these are information placeholders with the same semantics as marked nulls, but their incompleteness does not decrease in the presence of new knowledge: every interpretation of a persistent null is consistent with any additional knowledge.

Relational database domains with persistent nulls are simply relational database domains, where databases are populated by constants, nulls and persistent nulls. The latter values come from a countably infinite set PNull. We denote the elements of PNull using the symbol $\top$ with subscripts. For a database d, we denote by $\mathsf{PNull}(d)$ the set of persistent nulls appearing in it.

A P-*valuation* is a partial function v from $\mathsf{Null} \cup \mathsf{PNull}$ to Const. We denote by $v(d)$ the database obtained from d by replacing each element e of $\mathsf{Null} \cup \mathsf{PNull}$ with $v(e)$, if this is defined. We introduce P-valuations only as a means to define the semantics of relational database domains with persistent nulls; such P-valuations, however, do not provide additional knowledge, as the incompleteness of persistent nulls cannot be reduced. Indeed, we consider again the monoid $\mathcal{V}$ of partial valuations introduced in Section 4, which is additional to relational database domains with persistent nulls.

In what follows, we will only use relational database domains with persistent nulls as target, and we focus on open-world semantics as before. For simplicity, we use the collapsing semantics of P-valuations, which is defined below and is equivalent to the mixed one.

Definition 8. *The* collapsing application *of a* P*-valuation* v *to a* k*-ary table* T *is the* k*-ary table* $v(T)$ *such that, for every tuple* $\bar{t} \in \mathsf{Const}^k$ *we have:*

$$\#\big(v(T), \bar{t}\,\big) = \max\big\{\, \#(R, \bar{u}) \mid v(\bar{u}) = \bar{t}\,\big\}$$

The corresponding OWA *semantics of incompleteness is defined as follows:*

$$[\![d]\!]^{\text{OWA}} = \{d' \textit{ complete} \mid d' \supseteq v(d),\ v \textit{ is a } \mathsf{P}\textit{-valuation}\}$$

With all of this in place, we can capture information-based certain answers while still preserving knowledge from partial valuations.

Below, we show that the valuation-preserving certain answers with persistent nulls are as informative as information-based certain answers, and more informative than valuation-preserving certain answers without persistent nulls.

Proposition 5. *Let* $\mathbf{R}$ *be a relational database domain, and let* $\mathbf{P}$ *be a relational database domain with persistent nulls, such that* $\mathbb{C}_{\mathbf{R}} = \mathbb{C}_{\mathbf{P}}$*. Let* q *be a query from* $\mathbf{R}$ *to* $\mathbf{R}$*, and let* q' *be a query identical to* q*, but from* $\mathbf{R}$ *to* $\mathbf{P}$*. Then, for every* $d \in \mathbb{I}_{\mathbf{R}}$*, all of the following hold:*

(a) $\mathsf{cert}_{\square}(q,d) \equiv_{\mathbf{P}} \mathsf{cert}_{\mathcal{V}}(q',d)$*;*

(b) $\mathsf{cert}_{\mathcal{V}}(q,d) \preceq^{\mathcal{V}}_{\mathbf{P}} \mathsf{cert}_{\mathcal{V}}(q',d)$*.*

Since we are now able to capture information-based certain answers, the valuation-preserving certain answers will not be empty. As an application, we look at a class UCQ-$\mathcal{F}$ of value-inventing queries defined by unions of conjunctive queries with function application.

5.1 Query Answering for UCQ-$\mathcal{F}$

Let $\mathcal{F}$ be a set of functions where each function f of arity k is from Const^k to Const.

Definition 9. *The language* UCQ-$\mathcal{F}$ *of* union of conjunctive queries with function application *is defined by the following grammar:*

$$q := R \mid q \times q \mid q \uplus q \mid \pi_{\bar{\alpha}}(q) \mid \sigma_{\alpha_i=\alpha_j}(q) \mid \mathsf{Apply}_{\bar{\alpha}}(f, q) \quad \textit{with } f \in \mathcal{F}$$

where α_i and α_j are attributes, and $\bar{\alpha}$ is a tuple of attributes.

All of the operations above, except Apply*, are defined in the standard way (Console, Guagliardo, and Libkin 2019). The semantics of $q' = \mathsf{Apply}_{\alpha_1,\dots,\alpha_k}(f, q)$, for every complete relational database d and for every tuple $\bar{t}$ of the same arity as q, is defined as follows:*

$$\#\big(q'(d), (\bar{t}, t)\big) = \begin{cases} \#\big(q(d), \bar{t}\,\big) & \text{if } t = f\big(\bar{t}[\alpha_1], \dots, \bar{t}[\alpha_k]\big) \\ 0 & \text{otherwise} \end{cases}$$

While the query language UCQ-$\mathcal{F}$ is conceptually quite simple, computing the valuation-preserving certain answers for UCQ-$\mathcal{F}$ queries is intractable even if $\mathcal{F}$ consists of just one unary function.

Proposition 6. *There is a unary function f such that computing the $\{\varepsilon\}$-preserving certain answer and the valuation-preserving certain answer to Boolean queries in* UCQ-$\{f\}$*, on relational databases with or without persistent nulls, is coNP-hard in data complexity.*

For UCQ-$\mathcal{F}$ queries, the standard SQL evaluation is polynomial, so it cannot compute any knowledge-preserving notion of certain answers, which may result in counter-intuitive results on database with nulls. An approach to approximate certain answers when they are intractable is to build a query evaluation algorithm with correctness guarantees that runs in polynomial time.

Definition 10. *Let* $\mathbf{S}$ *and* $\mathbf{T}$ *be database domains, and $\mathcal{Q}$ be a query language. A query evaluation algorithm* $\mathsf{Eval}\colon \mathcal{Q} \times \mathbb{I}_{\mathbf{S}} \to \mathbb{I}_{\mathbf{T}}$ *has* $\mathcal{A}$-preserving correctness guarantees *for $\mathcal{Q}$ if, for every query $q \in \mathcal{Q}$, all of the following hold:*

(a) $\mathsf{Eval}(q, x) \preceq^{\mathcal{A}}_{\mathbf{T}} \mathsf{cert}_{\mathcal{A}}(q, x)$ *for every $x \in \mathbb{I}_{\mathbf{S}}$, and*

(b) $\mathsf{Eval}(q, c) = q(c)$ *for every $c \in \mathbb{C}_{\mathbf{S}}$.*

In other words, a query evaluation algorithm with correctness guarantees produces answers that are consistent (w.r.t. the $\mathcal{A}$-preserving information ordering on the target domain) with the query answers on incomplete databases, and equal to them on complete databases.

In order to obtain a polynomial-time algorithm, we extend the idea of naive evaluation on relational database domains based on a free algebra of terms.

5.2 Relational Databases over Free Algebra

We consider a free algebra of terms $\mathcal{T}$, where the basis of $\mathcal{T}$ is the set $\mathsf{Const} \cup \mathsf{Null}$ of constants and (marked) nulls, and the operations are functions from $\mathcal{T} \times \cdots \times \mathcal{T}$ to $\mathcal{T}$. The set of such functions is denoted by Γ. Then, a relational database is over $\mathcal{T}$ if it is populated with elements of $\mathcal{T}$.

Definition 11. *An* interpretation $\cdot^{\mathcal{I}}$ *of* Γ *over* Const *associates each $\gamma \in \Gamma$ of arity k with a function $\gamma^{\mathcal{I}}\colon \mathsf{Const}^k \to \mathsf{Const}$. The* grounding *of $\mathcal{T}$ under $\mathcal{I}$ is the function* grd *that maps each $t \in \mathcal{T}$ to an element of* $\mathsf{Const} \cup \mathsf{Null} \cup \mathsf{PNull}$ *as follows:*

$$\mathsf{grd}(t) = \begin{cases} t & \text{if } t \in \mathsf{Const} \cup \mathsf{Null} \\ \gamma^{\mathcal{I}}(c_1, \dots, c_k) & \text{if } t = \gamma(t_1, \dots, t_k) \text{ and } c_i = \mathsf{grd}(t_i) \in \mathsf{Const} \text{ for every } i \in \{1, \dots, k\} \\ \top_{\mathsf{grd}(t)} & \text{otherwise} \end{cases}$$

This naturally extends tables and relational databases over $\mathcal{T}$. The grounding of a table T is the table $\mathsf{grd}(T)$ with persistent nulls such that, for every tuple $\bar{t}$ of the same arity as T, we have:

$$\#\big(\mathsf{grd}(T), \bar{t}\big) = \sum_{\bar{u}\colon \mathsf{grd}(\bar{u}) = \bar{t}} \#(T, \bar{u})$$

Finally, the grounding of a relational database d over $\mathcal{T}$ is the database $\mathsf{grd}(d)$ of the same schema such that, for every table name R, we have $R^{\mathsf{grd}(d)} = \mathsf{grd}(R^d)$.

A relational database domain $\mathbf{D}$ is over the free algebra $\mathcal{T}$ if every database in $\mathbb{I}_{\mathbf{D}}$ is over $\mathcal{T}$ and its grounding belongs to $\mathbb{C}_{\mathbf{D}}$; in addition, $[\![\cdot]\!]_{\mathbf{D}}$ uses OWA with the collapsing semantics of valuations as defined below.

Definition 12. *For every valuation $v\colon \mathsf{Null} \to \mathsf{Const}$ and for every term $t = \gamma(t_1, \dots, t_k) \in \mathcal{T}$, we let $v(t)$ denote the term $\gamma\big(v(t_1), \dots, v(t_k)\big)$.*

The collapsing application *of v to a k-ary table T over $\mathcal{T}$ is the k-ary table $v(T)$ such that, for every $\bar{t} \in \mathcal{T}^k$, we have:*

$$\#\big(v(T), \bar{t}\big) = \max\big\{\, \#(R, \bar{u}) \mid v(\bar{u}) = \bar{t} \big\}$$

The corresponding OWA *semantics of incompleteness is defined as follows:*

$$[\![d]\!]^{\mathrm{OWA}} = \{d' \textit{ complete} \mid d' \supseteq v(d),\ v \textit{ is a valuation}\}$$

We will now use the relational database domain over $\mathcal{T}$ to define naive evaluation for value-inventing queries.

5.3 Approximation Algorithms for UCQ-$\mathcal{F}$

For the rest of this section, we consider the free algebra of terms $\mathcal{T}$ defined as above, with a set Γ of operations whose interpretation $\mathcal{I}$ over Const is assumed to be computable in polynomial time. We take $\mathcal{F} = \{\gamma^{\mathcal{I}} \mid \gamma \in \Gamma\}$ as the set of functions for UCQ-$\mathcal{F}$ queries.

Definition 13. *The* interpreted naive evaluation *of a* UCQ-$\mathcal{F}$ *query q is the query $q_{\textit{i-naive}}$ from incomplete relational databases to incomplete relational databases over $\mathcal{T}$, where $q_{\textit{i-naive}}$ is the standard naive evaluation for relational algebra (i.e., nulls are treated as new constants) and such that, for every relational database d and for every tuple $\bar{t}$ of appropriate arity, we have:*

- *for $q = \mathsf{Apply}_{\alpha_1,\dots,\alpha_k}(\gamma^{\mathcal{I}}, q')$; $\#\big(q_{\textit{i-naive}}(d), (\bar{t}, t)\big)$ is*

$$\begin{cases} \#\big(q'_{\textit{i-naive}}(d), \bar{t}\,\big) & \text{if } t = \gamma(\bar{t}[\alpha_1], \dots, \bar{t}[\alpha_k]) \\ 0 & \text{otherwise} \end{cases}$$

- *for* $q = \sigma_{\alpha_i=\alpha_j}(q')$*;* $\#\big(q_{\textit{i-naive}}(d), \bar{t}\big)$ *is*

$$\begin{cases} \#(q'_{\textit{i-naive}}(d), \bar{t}) & \textit{if } \mathsf{grd}(\bar{t}[\alpha_i]) = \mathsf{grd}(\bar{t}[\alpha_j]) \\ 0 & \textit{otherwise} \end{cases}$$

Informally, the interpreted naive evaluation behaves like the classical naive evaluation for relational algebra, except that it interprets the free algebra terms when checking for equality in selections.

Proposition 7. *For* UCQ-$\mathcal{F}$ *query* q *and for every relational database* d*, the interpreted naive evaluation of* q *can be computed in polynomial time in the size of* d*.*

We have built a polynomial-time evaluation algorithm for UCQ-$\mathcal{F}$ queries on relational database domains over $\mathcal{T}$. In addition, by means of grounding, we can provide an evaluation algorithm with correctness guarantees.

Theorem 4. *The grounding of the interpreted naive evaluation is a query evaluation algorithm with correctness guarantees for* UCQ-$\mathcal{F}$*. Thus, for every query* $q \in$ UCQ-$\mathcal{F}$ *and for every relational database* d*, we have:*

(a) $\mathsf{grd}\big(q_{\textit{i-naive}}(d)\big) \preceq^{\mathcal{V}}_{\mathbf{P}} \mathsf{cert}_{\mathcal{V}}(q, d)$*; and*

(b) *if* d *is complete, then* $\mathsf{grd}\big(q_{\textit{i-naive}}(d)\big) = q(d)$.

We conclude this section with a fully worked-out example of how UCQ-$\mathcal{F}$ queries are evaluated and grounded.

Example 5. Consider a database D with a relation $R = \{(3,0,1,2),\ (1,\bot_1,1,\bot_1), (2,\bot_2,\bot_2,2)$ over attributes A, B, C, D, and the following value inventing query q':

$$\pi_{(A+B);(C+D)}\big(\mathsf{Apply}_{(C,D)}(+, \mathsf{Apply}_{(A,B)}(+, R))\big)$$

where $+(c, c') = c + c'$ for every $c, c' \in \mathsf{Const}$. The naive evaluation of q' on D gives the following answers:

$q'_{\text{i-naive}}(D) = q'_{\text{naive}}(D) =$

$A+B$	$C+D$
$+(3,0)$	$+(1,2)$
$+(1,\bot_1)$	$+(1,\bot_1)$
$+(2,\bot_2)$	$+(\bot_2,2)$

The naive evaluation of $q = \sigma_{(A+B)=(C+D)}\big(q'\big)$ on D and its grounding are:

$q_{\text{naive}}(D)$

$A+B$	$C+D$
$+(1,\bot_1)$	$+(1,\bot_1)$

$\mathsf{grd}\big(q_{\text{naive}}(D)\big)$

$A+B$	$C+D$
$\top_i$	$\top_i$

These return unsatisfactory answers, because they miss some complete tuples. Naive evaluation cannot capture the fact that $+(0,3) = +(1,2) = 3$, as it does not interpret the function. On the other hand, the interpreted naive evaluation and its grounding are:

$q_{\text{i-naive}}(D)$

$A+B$	$C+D$
$+(3,0)$	$+(1,2)$
$+(1,\bot_1)$	$+(1,\bot_1)$

$\mathsf{grd}\big(q_{\text{i-naive}}(D)\big)$

$A+B$	$C+D$
3	3
$\top_j$	$\top_j$

The interpreted naive evaluation is able to capture all complete tuples. It recognizes that constant terms such as $+(3,0)$ and $+(1,2)$ are equal. Moreover, for this query, the interpreted-naive evaluation captures more tuples than the certain answers with nulls, but strictly less tuples than the valuation-preserving certain answers on the relational database domain with persistent nulls:

$\mathsf{cert}_{\bot}(q, D)$

$A+B$	$C+D$
3	3

$\mathsf{cert}_{\mathcal{V}}(q, D)$

$A+B$	$C+D$
3	3
$\top_j$	$\top_j$
$\top_k$	$\top_k$

The valuation-preserving certain answer captures more tuples, because it is able to recognize that $+(2,\bot_2)$ and $+(\bot_2,2)$ are equal for every valuation.

We have been able to build a polynomial query evaluation algorithm with correctness guarantee for the applied union of conjunctive query language UCQ-$\mathcal{F}$. Moreover as illustrated in Example 5, there exist queries where the interpreted naive evaluation algorithm is a strict improvement from the certain answers with nulls: it returns more certain tuples.

6 Conclusions

A big obstacle in further developing areas where AI techniques, in particular logic-based ones, interact with data management is understanding notions of query answers over incomplete data. Indeed, applications such as OBDA, data integration, data exchange combine reasoning and query answering tasks, but in the end need to rely on database technology to produce answers. This technology is to be applied in settings where data is naturally incomplete, by the very means of its construction. This in turn severely limits the applicability of the techniques: one typically resorts to conjunctive queries, or their unions, or a handful of very closely related classes where we have an understanding of correctness of query answers and an arsenal of techniques for computing those answers.

However, if one looks at real life database queries (e.g., in standard benchmarks, as those produced by the TPC), there are very few queries from the classes for which we have query answering techniques in the presence of incomplete data. Real-life queries differ in the most basic semantics of the underlying data model, and in their features, as crucially they invent new values that form part of the output. For those, we lacked notions of correctness, or certainty, of query answering.

Our goal here was to remedy this situation by first providing a general framework explaining what correctness is, and second by showing how it can be applied in some cases that go well beyond queries that we had known how to handle. We have done so for the prevalent bag semantics, and for queries that can invent new values by means, for example, of arithmetic functions.

The most pressing next question to us is to extend these techniques to aggregate queries that can produce new values by applying arithmetic functions to entire columns in relations. These are extremely common in applications, as witnessed again by benchmark queries, and theoretical literature still offers no insight into the notions of correctness of query answering with aggregates over incomplete data. We expect techniques presented here to open up new ways of attacking this long standing open problem.

Acknowledgements

Work partially supported by EPSRC grants M025268 and N023056, EPSRC Centre for Doctoral Training in Data Science, a grant from the Foundation Sciences Mathématiques de Paris under the FSMP Chairs program, and the Royal Society through a Wolfson Research Merit Award.

References

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

Amendola, G., and Libkin, L. 2018. Explainable certain answers. In *IJCAI*, 1683–1690. ijcai.org.

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

Bertossi, L. E. 2011. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

Bienvenu, M., and Ortiz, M. 2015. Ontology-mediated query answering with data-tractable description logics. In *Reasoning Web*, volume 9203 of *Lecture Notes in Computer Science*, 218–307. Springer.

Buneman, P.; Jung, A.; and Ohori, A. 1991. Using powerdomains to generalize relational databases. *Theor. Comput. Sci.* 91(1):23–55.

Calì, A.; Lembo, D.; and Rosati, R. 2003. Query rewriting and answering under constraints in data integration systems. In *IJCAI*, 16–21. Morgan Kaufmann.

Codd, E. F. 1979. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst.* 4(4):397–434.

Console, M.; Guagliardo, P.; Libkin, L.; and Toussaint, E. 2020. Coping with incomplete data: Recent advances. In *PODS*, 33–47. ACM.

Console, M.; Guagliardo, P.; and Libkin, L. 2017. On querying incomplete information in databases under bag semantics. In *IJCAI*, 993–999. ijcai.org.

Console, M.; Guagliardo, P.; and Libkin, L. 2019. Fragments of bag relational algebra: Expressiveness and certain answers. In *ICDT*, volume 127 of *LIPIcs*, 8:1–8:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Franconi, E., and Tessaris, S. 2012. On the logic of SQL nulls. In *AMW*, volume 866 of *CEUR Workshop Proceedings*, 114–128. CEUR-WS.org.

Grahne, G. 1991. *The Problem of Incomplete Information in Relational Databases*, volume 554 of *Lecture Notes in Computer Science*. Springer.

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

Guagliardo, P., and Libkin, L. 2016. Making SQL queries correct on incomplete databases: A feasibility study. In *PODS*, 211–223. ACM.

Hernich, A., and Kolaitis, P. G. 2017. Foundations of information integration under bag semantics. In *LICS*, 1–12. IEEE Computer Society.

Imielinski, T., and Jr., W. L. 1984. Incomplete information in relational databases. *J. ACM* 31(4):761–791.

Jr., W. L. 1979. On semantic issues connected with incomplete information databases. *ACM Trans. Database Syst.* 4(3):262–296.

Jr., W. L. 1984. On relational algebra with marked nulls. In *PODS*, 201–203. ACM.

Kontchakov, R.; Lutz, C.; Toman, D.; Wolter, F.; and Zakharyaschev, M. 2011. The combined approach to ontology-based data access. In *IJCAI*, 2656–2661. IJCAI/AAAI.

Lenzerini, M. 2002. Data integration: A theoretical perspective. In *PODS*, 233–246. ACM.

Libkin, L. 1995. A semantics-based approach to design of query languages for partial information. In *Semantics in Databases*, volume 1358 of *Lecture Notes in Computer Science*, 170–208. Springer.

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

Libkin, L. 2016a. Certain answers as objects and knowledge. *Artif. Intell.* 232:1–19.

Libkin, L. 2016b. Sql's three-valued logic and certain answers. *ACM Trans. Database Syst.* 41(1):1:1–1:28.

Poggi, A.; Lembo, D.; Calvanese, D.; Giacomo, G. D.; Lenzerini, M.; and Rosati, R. 2008. Linking data to ontologies. *J. Data Semantics* 10:133–173.

Rounds, B. 1991. Situation-theoretic aspects of databases. In *Situation Theory and Applications, Volume 2*, volume 26 of *CSLI Lecture Notes*. CSLI Publications. 229–256.

Transaction Processing Performance Council. 2014. *TPC Benchmark™ H Standard Specification*. Revision 2.17.1.

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