Certain Answers Meet Zero–One Laws

Leonid Libkin
University of Edinburgh
libkin@inf.ed.ac.uk

ABSTRACT

Query answering over incomplete data invariably relies on the standard notion of certain answers which gives a very coarse classification of query answers into those that are certain and those that are not. Here we propose to refine it by *measuring* how close an answer is to certainty.

This measure is defined as the probability that the query is true under a random interpretation of missing information in a database. Since there are infinitely many such interpretations, to pick one at random we adopt the approach used in the study of asymptotic properties and 0–1 laws for logical sentences, and define the measure as the limit of a sequence. We show that in the standard model of missing data, the 0–1 law is observed: this limit always exists and can be only 0 or 1 for a very large class of queries. Thus, query answers are either almost certainly true, or almost certainly false. We prove that almost certainly true answers are precisely those returned by the naïve evaluation of the query. When databases satisfy constraints, the measure is defined as the conditional probability of the query being true if the constraints are true. This too is defined as a limit, and we prove that it always exists, can be an arbitrary rational number, and is computable. For some constraints, such as functional dependencies, the 0–1 law continues to hold.

As another refinement of the notion of certainty, we introduce a comparison of query answers: an answer with a larger set of interpretations that make it true is better. We identify the precise complexity of such comparisons, and of finding sets of best answers, for first-order queries.

ACM Reference Format:
Leonid Libkin. 2018. Certain Answers Meet Zero–One Laws. In *PODS'18: 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3196959.3196983

1 INTRODUCTION

When a database has incomplete or uncertain information, the standard approach to answering queries against it is to compute *certain answers*. The exact definition depends on the nature of incompleteness and its precise semantics, but the idea is invariably the same: these are the answers that do not depend on how incomplete data (e.g., null values in databases) is interpreted. This approach permeates practically all other scenarios where a database is not fully specified, such as data integration [30], data exchange [3, 19], ontology-based data access (OBDA) [9, 35], consistent query answering [8], and others.

In many of these scenarios, finding certain answers is a hard task computationally, except for some restricted classes of queries. Nonetheless, the central notion around which query answering revolves is that of certain answers. They are viewed as the ultimate goal, either to be computed or approximated, and everything that is not certain is dismissed as a candidate for an answer. Occasionally one also looks at possible answers, but in terms of classifying query answers, this is basically all: there are no finer notions.

Since precise computation of certain answers is hard, applications that must evaluate queries efficiently are bound to do something different. A typical example is *naïve evaluation* for databases with null values which simply treats nulls as regular database entries. This is very common in data integration, exchange, and OBDA scenarios, where queries are directly applied to databases with nulls [3, 9, 19, 30, 35], even though we know that in general naïve evaluation cannot produce certain answers. What we do not know, however, is how far the result of naïve evaluation is from certain answers. In fact we do not even have a framework in which we can answer such questions.

Our goal therefore is to provide a framework for reasoning about measures of certainty of query answers, to enable a finer classification of answers to queries over incomplete data. We want to be able to say *how certain* an answer is, or whether one answer is *better* than another. We now illustrate the usefulness of such notions by an example that also introduces the key concepts needed here.

Example: measuring and comparing certainty. Consider a decision support scenario where one tries to infer information about customers' habits. Suppose that by querying or perhaps restructuring some data, we found information about products that customers buy from two suppliers, and stored that information in relations R_1 and R_2. As is often the case when data is obtained from multiple sources [3, 30], some entries are missing (i.e., they are nulls). Below we show part of a database D that gives portions of these relations relevant to customers c_1 and c_2:

R_1

customer	product
c_1	$\perp_1$
c_2	$\perp_1$
c_2	$\perp_2$

R_2

customer	product
c_1	$\perp_2$
c_2	$\perp_1$
$\perp_3$	$\perp_1$

These are relations with *marked* (or labeled) nulls [1, 27], denoted by the symbol $\perp$. This model is very common, and more expressive than SQL's primitive model of a single null value. Nulls mean that a value exists but is unknown at present; the occurrence of the same null in different positions indicates that it is the same

PODS'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4706-8/18/06…$15.00
https://doi.org/10.1145/3196959.3196983

value that occurs there, even if we do not know what it is. For instance, we know that c_1 and c_2 buy the same product, $\perp_1$, from both suppliers, but we do not know what this product is. The semantics of such databases is given by means of *valuations*, i.e., mappings v that assign values to nulls. We write $v(D)$ for the result of applying v to D, i.e., replacing each null $\perp$ with $v(\perp)$.

Certain answers to a query Q returning a set of tuples are defined as

$$\Box(Q, D) = \{\bar{a} \mid v(\bar{a}) \in Q(v(D)) \text{ for each valuation } v\}.$$

This is the definition from [33], sometimes called *certain answers with nulls*. In the literature it is more common to look at certain answers defined as $\bigcap_v Q(v(D))$, with the intersection taken over all valuations [27], but this is simply the restriction of $\Box(Q, D)$ to tuples without nulls [32]. We choose the more permissive definition of [33], as it has a number of advantages: for instance, if a query Q returns relation R_1, then $\Box(Q, D) = R_1$, while the intersection-based certain answers will return $\emptyset$.

Consider a query Q asking for products that customers bought only from the first supplier, i.e., $Q(x, y) = R_1(x, y) \wedge \neg R_2(x, y)$ in logical notation. Then $\Box(Q, D) = \emptyset$. Evaluating Q naïvely on D produces two tuples $(c_1, \perp_1)$ and $(c_2, \perp_2)$ which are not certain answers: if $v(\perp_1) = v(\perp_2)$, then the condition $v(\bar{a}) \in Q(v(D))$ fails for them. But in a large database with many products, and customers placing many orders, it is not very likely that $\perp_1$ and $\perp_2$ refer to the same product. It might happen of course – that is why these tuples are not certain – but for most of valuations of nulls, they will be in the output. Thus, $(c_1, \perp_1)$ and $(c_2, \perp_2)$ are likely, but not certain, answers to Q.

Is one of these tuples a better answer than the other? We can easily check that every valuation v for which $(c_1, v(\perp_1)) \in Q(v(D))$ also satisfies $(c_2, v(\perp_2)) \in Q(v(D))$, but the converse fails (because $v(\perp_3)$ could be c_1). Thus, there are strictly more valuations supporting $(c_2, \perp_2)$ as an answer; in fact no other tuple has more valuations supporting it, which indicates that $(c_2, \perp_2)$ is not only a likely answer, but also the *best* among likely answers.

Finally, assume that the customer field determines the product field. Then $v(\perp_1) = v(\perp_2)$ for all valuations v, and thus every $Q(v(D))$ is empty. Even though without constraints we could say that tuples $(c_1, \perp_1)$ and $(c_2, \perp_2)$ are likely answers to Q, with the constraint we know with certainty that they will not be answers. □

This example shows that there is more to answering queries over incomplete databases than just finding certain answers. We can also:

- ask *how likely* a tuple is to be an answer to a query;
- *compare* tuples and say which is likelier – or most likely – to appear in the query answer;
- measure this likelihood under *constraints*.

However, we lack a framework that can be used to analyze such measures of certainty. Our main goal therefore is to present such a framework. We now outline its key ideas and the main results.

Measuring certainty

For illustration purposes, assume that we have a Boolean (yes/no) query Q over incomplete relational databases D. The key notion is that of a *support* $\text{Supp}(Q, D)$ of Q over D which is the set of all valuations v such that Q is true in $v(D)$.

The idea is to consider a random valuation v of nulls, and compute the probability that Q is true under it, i.e., the probability that v is in the support of Q over D. There are infinitely many valuations though, so we cannot pick one uniformly at random. But instead we can use the approach from the study of 0–1 laws and asymptotic behavior of logical properties, where one tries to define how likely a randomly chosen structure is to satisfy a given property [17, 31, 39]. Assume some enumeration $\{c_1, c_2, \ldots\}$ of non-null elements, and define $\text{V}^k(D)$ as the set of valuations that take values in $\{c_1, \ldots, c_k\}$, and $\text{Supp}^k(Q, D)$ as the restriction of $\text{Supp}(Q, D)$ to the same set. Both are finite, and we can thus define

$$\mu^k(Q, D) = \frac{|\text{Supp}^k(Q, D)|}{|\text{V}^k(D)|}$$

as the probability that a randomly picked valuation v from $\text{V}^k(D)$ witnesses that Q is true in $v(D)$. Then we look at the asymptotic behavior of this sequence, i.e., $\mu(Q, D) = \lim_{k\to\infty} \mu^k(Q, D)$, if such a limit exists. A case of particular interest is when $\mu(Q, D) = 1$. Then Q is almost certainly true in D, i.e., true for almost all valuations of nulls.

The questions we address are the following.

- What values can $\mu(Q, D)$ take? We prove a 0–1 law: its values could be only 0 or 1. In other words, a query is either almost certainly true or almost certainly false. This holds for a very large class of queries: the only condition we need is a standard database notion of genericity (in particular, this result is quite different from 0–1 laws in logic, as it holds for much larger classes of queries).
- When $\mu(Q, D) = 1$? This happens if and only if the naïve evaluation of Q on D returns *true*. Thus, naïve evaluation is not so arbitrary after all: it checks if the query is almost certainly true.
- What happens when databases satisfy constraints? Above we assumed that all valuations are equally likely, but this is not the case when databases are required to satisfy a set Σ of constraints (e.g., keys and foreign keys or, more generally, functional and inclusion dependencies). Then the right measure is the *conditional probability* $\mu(Q|\Sigma, D)$ of Q being true if Σ is true. We show that such $\mu(Q|\Sigma, D)$, defined as an appropriate limit, always exists, but need not be 0 or 1: its values could be arbitrary rational numbers in $[0, 1]$. However, sometimes the 0–1 law is true even under constraints, for example, when Σ contains only functional dependencies.

Comparing answers

Now we look at queries Q that return sets of tuples, and extend the notion of support by letting $\text{Supp}(Q, D, \bar{a})$ be the set of valuations v such that $v(\bar{a}) \in Q(v(D))$. We use this notion to ask whether $\bar{a}$ is *better* than some other tuple $\bar{b}$ as an answer to Q, or whether $\bar{a}$ is among the *best* possible answers.

If $\text{Supp}(Q, D, \bar{a}) \subseteq \text{Supp}(Q, D, \bar{b})$, then $\bar{b}$ has at least as much support as $\bar{a}$, and thus is at least as likely to be an answer. If $\text{Supp}(Q, D, \bar{a}) \subset \text{Supp}(Q, D, \bar{b})$, then $\bar{b}$ has more support than $\bar{a}$, and thus is more likely to be an answer. Tuples whose support is

maximal with respect to inclusion give us the set of *best* answers. Best answers always exist, unlike certain answers that could be empty. The questions we address are the following.

- How hard are these comparisons? For queries in relational algebra/calculus (and more generally, for queries with polynomial time data complexity), the problems mentioned in the previous paragraph are all within the second level of the polynomial hierarchy, in fact in $\Sigma_2^p \cap \Pi_2^p$. More precisely, checking whether $\bar{b}$ is at least as good as $\bar{a}$ is coNP-complete, checking whether $\bar{b}$ is strictly better than $\bar{a}$ is DP-complete, and identifying the set of best tuples is $\mathrm{P}^{\mathrm{NP}[\log n]}$-complete (we recall definitions of these classes in the next section).
- When are these problems tractable? We show that they are for unions of conjunctive queries. At first this does not sound surprising since we know that finding certain answers for unions of conjunctive queries is easy by naïve evaluation [1, 27]. However, here naïve evaluation is of no help, and the algorithms are of quite a different nature.

The immediate consequence of these new ways of looking at answering queries over incomplete databases is a framework in which both quantitative and qualitative comparisons of query answers could be made. Already early applications of the framework vindicate to an extent the often used naïve evaluation: it is not totally arbitrary, but rather it produces answers with very good probabilistic guarantees. We think that the added flexibility of these notions will be useful in addressing questions that have been hard to answer using only the rigid notion of certain answers, for instance, questions about the quality of approximate algorithms for query answering over incomplete data.

Organization. In Section 2 we recall background concepts, including the basics of 0–1 laws. In Section 3 we define the measure of certainty of answers and prove two versions of the 0–1 law. In Section 4 we look at databases satisfying constraints, prove the convergence result for the measure defined as the conditional probability, and study cases when the 0–1 law can be recovered. In Section 5 we show how to compare tuples and identify best tuples, study the complexity of associated problems and relate these notions to quantitative measurements of certainty. In Section 6 we provide directions for future research based on this new framework.

2 PRELIMINARIES

Incomplete databases

We consider incomplete databases with nulls interpreted as missing information. Below we recall definitions that are standard in the literature [1, 27, 42]. Databases are populated by two types of elements: *constants* and *nulls*, coming from countably infinite sets denoted by Const and Null, respectively. Nulls are denoted by $\perp$, sometimes with sub- or superscripts. If nulls can repeat in a database, they are referred to as *marked*, or labeled, nulls; otherwise one speaks of *Codd nulls*, which are the usual way of modeling SQL's nulls. Marked nulls are standard in applications such as data integration and exchange and OBDA [3, 9, 30], and are more general than Codd nulls; hence we use them here.

A relational schema is a set of relation names with associated arities. In an incomplete relational instance D, each k-ary relation symbol R from the vocabulary is interpreted as a k-ary relation over $\mathsf{Const} \cup \mathsf{Null}$. In other words, such a relation is a finite subset of $(\mathsf{Const} \cup \mathsf{Null})^k$.

Slightly abusing notation (as it will never lead to confusion here) we will call it R as well.

The sets of constants and nulls that occur in a database D are denoted by $\mathsf{Const}(D)$ and $\mathsf{Null}(D)$, respectively. The *active domain* of D is $\mathrm{adom}(D) = \mathsf{Const}(D) \cup \mathsf{Null}(D)$. If D has no nulls, we say that it is *complete*.

Valuations and query answering

A *valuation* v on a database D is a map $v : \mathsf{Null}(D) \to \mathsf{Const}$ that assigns constant values to nulls occurring in the database. By $v(D)$ we denote the result of replacing each null $\perp$ with $v(\perp)$ in D. The *semantics* $[\![D]\!]$ of an incomplete database D is the set of all complete databases it can represent, i.e., $\{v(D) \mid v \text{ is a valuation}\}$. This is known as the closed-world semantics of incompleteness [36]; we shall also comment on the open-world semantics in the paper. We write $\mathrm{range}(v)$ for the range of valuation v, i.e., the set $v(\mathsf{Null}(D)) \subset \mathsf{Const}$. The set of all valuations defined on D is denoted by $\mathsf{V}(D)$.

An *query* of arity m (or an m-ary query) is a map that associates with a database D a subset of m-tuples over its elements, i.e., a subset of $\mathrm{adom}(D)^m$. Queries in standard languages such as relational algebra, calculus, datalog, etc., are such; in this initial investigation, we do not consider queries that can invent new values, i.e., return constants that are not in the active domain. A *Boolean* query is a query of arity 0. There is only tuple of arity zero, namely the empty tuple (). As usual, we associate *false* with the empty set $\emptyset$, and *true* with the set $\{()\}$ containing the empty tuple. For Boolean queries, we can write alternatively $Q(D) = \mathit{true}$ or $D \models Q$.

An m-tuple $\bar{a}$ over $\mathrm{adom}(D)$ is a *certain answer* to an m-ary query Q over D if $v(\bar{a}) \in Q(v(D))$ for all valuations v. The set of all certain answers to Q over D is denoted by $\Box(Q, D)$.

If $\bar{a}$ contains only constants, this means that $\bar{a} \in Q(D')$ for all $D' \in [\![D]\!]$. This is the notion one sees most commonly in the literature, but we prefer the above definition, due to [33], since it does not impose an artificial restriction that only constants be present in the answer. For instance, if Q returns a relation R in a database, then $\Box(Q, D) = R$, which is more natural than restricting R to tuples without nulls. Note also that if Q is a Boolean query, then $\Box(Q, D) = \mathit{true}$ iff $Q(D') = \mathit{true}$ for all $D' \in [\![D]\!]$.

Supports for answers

Given a query Q, a database D, and a tuple $\bar{a}$ over $\mathrm{adom}(D)$, define the *support* of $\bar{a}$ being an answer to Q on D as the set of all valuations that witness it:

$$\mathrm{Supp}(Q, D, \bar{a}) \;=\; \{v \in \mathsf{V}(D) \mid \; v(\bar{a}) \in Q(v(D))\}\,.$$

Supports thus measure how closely a tuple is to certainty. A tuple $\bar{a}$ is in $\Box(Q, D)$ iff $\mathrm{Supp}(Q, D, \bar{a}) = \mathsf{V}(D)$, i.e., the support includes all valuations. One can describe possible answers to queries via supports too; in that case, $\mathrm{Supp}(Q, D, \bar{a}) \neq \emptyset$.

Basics of 0–1 laws

Much of the discussion on measuring certainty will invoke comparisons with the study of 0–1 laws and asymptotic behavior of logical sentences. The key idea behind such a study is to pick randomly a relational structure, and compute the probability that a sentence φ in a logic is true in it. To explain this in a bit more detail, we use, for simplicity, graphs, i.e., structures $G = \langle V, E\rangle$ where V is a set of nodes and E is a set of edges. Since there are infinitely many finite graphs, we cannot pick one uniformly at random with a non-zero probability, but we can do so if the set from which nodes are drawn is finite. Thus, we let Gr_k be the set of all graphs whose nodes come from $\{1, \ldots, k\}$, and then, for a sentence φ of some logic, define:

$$\mu^k(\varphi) \;=\; \frac{|\{G \in \mathrm{Gr}_k \mid G \models \varphi\}|}{|\mathrm{Gr}_k|}$$

as the proportion of graphs from Gr_k that satisfy φ. It can also be interpreted as follows: for each pair $i, j \le k$, toss a coin and with probability $\frac{1}{2}$ put an edge (i, j); then $\mu^k(\varphi)$ is the probability that the resulting graph satisfies φ.

One is interested in the asymptotic behavior of the sequence $(\mu^k(\varphi))_{k\ge 0}$, i.e., in the limit $\mu(\varphi) = \lim_{k\to\infty} \mu^k(\varphi)$. In general, one can observe several kinds of behavior.

- ***0–1 law*** In this case $\mu(\varphi)$ exists and equals 0 or 1. If it equals 1, then φ is almost surely true, otherwise it is almost surely false. The celebrated theorem (and proof of it) by Fagin [17] states that first-order logic has the 0–1 law. Many extensions are known: for instance, the least fixed-point logic [10], the finite-variable logic [28], and some fragments of second-order logic [29] have the 0–1 law.
- ***Convergence*** For some logics, the sequence $(\mu^k(\varphi))_{k\ge 0}$ converges and the limit $\mu(\varphi)$ exists, but may not equal 0 or 1. Such a situation may occur if we restrict the class of graphs. For instance, consider graphs with an additional successor relation on their nodes, i.e., structures with two binary relations, of which one is a successor relation on the universe. Let φ state that there is an edge between the first and last elements of the successor relation; then $\mu(\varphi) = \frac{1}{2}$. In fact it is known that first-order logic over graphs with successor satisfies the convergence condition [34].
- ***Divergence*** Sometimes $(\mu^k(\varphi))_{k\ge 0}$ does not even converge. Consider, for example, property φ "the number of nodes is even". Then $\mu^k(\varphi)$ alternates between 0 and 1, and the limit does not exist.

One can also think of an alternative definition of the measure μ^k: instead, one defines a different quantity $\nu^k(\varphi)$ as the proportion of the number of isomorphism types of graphs, rather than graphs themselves, on $\{1, \ldots, k\}$ that satisfy φ. Even though $\mu^k(\varphi)$ and $\nu^k(\varphi)$ need not be the same, asymptotically these sequences behave in the same way, i.e., have the same limit, see [17].

Query languages

For many results in the paper we shall need only minimal assumptions on a query language, such as genericity of its queries (formally defined below in Section 3.1). For others, especially in Section 5, we shall restrict ourselves to *first-order* (FO) queries, or alternatively relational algebra/calculus queries, written here in the logical notation using Boolean connectives $\wedge, \vee, \neg$ and quantifiers $\exists, \forall$. The $\exists, \wedge$-fragment of FO is known as *conjunctive queries* (equivalently, select-project-join queries of relational algebra), and the $\exists, \wedge, \vee$-fragment is known as the *unions of conjunctive queries* (equivalently, select-project-join-union queries of relational algebra).

Complexity classes

In what follows, we shall need two classes in the second level of the polynomial hierarchy (in addition to the commonly used classes such as P, NP, coNP). Both of these contain NP and coNP, and are contained in $\Sigma_2^p \cap \Pi_2^p$.

The class DP consists of languages $L_1 \cap L_2$ where $L_1 \in$ NP and $L_2 \in$ coNP (or, equivalently, of differences of languages $L_1 - L_2$ with $L_1, L_2 \in$ NP). This class has appeared in database applications, for example in connection with the identification of the core in data exchange [18], or in the study of approximation of conjunctive queries [7].

The class $\mathrm{P}^{\mathrm{NP}[\log n]}$ consists of problems that can be solved in polynomial time with a logarithmic number (more precisely, $O(\log n)$ where n is the size of the input) of calls to an NP oracle [44]. Equivalently, it can be described as the class of problems solved in P with an NP oracle where calls to the oracle are done in parallel, i.e., independent of each other [11]. It is again not unheard of in database applications: for example, it was used to analyze the complexity of data exchange problems [4] and RDF query languages [5].

We shall also need the complexity class #P. It no longer contains decision problems, but rather *counting* problems, more specifically, counting numbers of accepting paths of an NP computation [6].

3 MEASUREMENTS OF CERTAINTY

The key idea, as explained earlier, is this. Given an incomplete database D, a query Q, and a tuple $\bar{a}$, pick a random valuation v of nulls, and calculate the probability of $v(\bar{a})$ being in $Q(v(D))$. We prove the 0–1 law result for query answers for arbitrary databases, and the convergence results for databases that satisfy constraints. But we start with a formal definition of naïve evaluation that uses the notion of *genericity* of queries.

3.1 Generic queries and naïve evaluation

Queries expressed in logic-based query languages are *generic*; essentially, they just manipulate data. This is a classical notion of database theory, often formulated as the data independence principle [1]: a query evaluation algorithm does not depend on particular data stored in a database, except perhaps a finite number of constants that could be mentioned explicitly in the query. A generic query thus commutes with permutations of the domain (where such permutations may leave some number of elements fixed).

Definition 1 (Generic query). *Given a finite set $C \subset$ Const, a query Q is called C-*generic *if for every bijection π : Const $\to$ Const such that $\pi(c) = c$ for all $c \in C$, and for every database D over* Const, *we have*

$$Q(\pi(D)) = \pi(Q(D)) \,.$$

A query is generic *if it is C-generic for some finite C.*

All queries definable in familiar logics such as first-order, its fixed-point extensions, second-order and its fragments, etc., are generic.

The notion of genericity lets us define naïve evaluation formally. In essence, it evaluates a query on an incomplete database as if nulls were distinct constants not occurring elsewhere in the database or the query. The concept of replacing nulls by such distinct constants is captured by the notion of bijective valuations.

Definition 2 (Bijective valuation). *For a database D and a finite set $C \subset \mathsf{Const}$, a valuation v on $\mathsf{Null}(D)$ is called C-*bijective *if it assigns to each $\perp \in \mathsf{Null}(D)$ a distinct constant that does not occur in $\mathsf{Const}(D)$ and C, i.e., if it is a bijective mapping and $\mathsf{range}(v)$ is disjoint from $\mathsf{Const}(D)$ and C.*

Next, we need an easy observation.

Proposition 1. *Let Q be a C-generic query, and let v, w be two C-bijective valuations. Then $v^{-1}(Q(v(D))) = w^{-1}(Q(w(D)))$.* □

This observation lets us define naïve evaluation formally.

Definition 3 (Naïve evaluation). *Let Q be a C-generic query. Then* naïve evaluation *of Q on a database D is defined as*

$$Q^{\text{naïve}}(D) \;=\; v^{-1}(Q(v(D)))$$

where v is any C-bijective valuation.

Consider, for example, a query on graphs $\varphi(x) = \exists y\, E(c, y) \wedge E(y, x)$ that looks for nodes x of distance 2 from node c. This query is $\{c\}$-generic. Suppose we have a graph G with edges (c, c'), $(c', \perp)$. To evaluate this naïvely, pick a valuation v that assigns a new constant c'' to $\perp$. Then $v(G) = \{(c, c'), (c', c'')\}$ and φ evaluated on $v(G)$ produces $\{c''\}$. Thus, φ evaluated naïvely on G returns $v^{-1}(\{c''\}) = \{\perp\}$, as expected.

3.2 0–1 law for generic queries

We now briefly recall the key idea of measuring the degree of certainty of answers. Recall that $\mathsf{Supp}(Q, D, \bar{a})$, the support of $\bar{a}$, is the set of all valuations v witnessing $v(\bar{a}) \in Q(v(D))$, and $\mathsf{V}(D)$ is the set of all valuations on D. We need to measure how likely a randomly chosen valuation from $\mathsf{V}(D)$ is to be in $\mathsf{Supp}(Q, D, \bar{a})$. For this, assume that the set of constants is enumerated as $\mathsf{Const} = \{c_1, c_2, \ldots\}$. Define $\mathsf{V}^k(D)$ as the set of valuations whose range is contained among the first k elements of this enumeration, i.e., $\mathsf{V}^k(D) = \{v \in \mathsf{V}(D) \mid \mathsf{range}(v) \subseteq \{c_1, \ldots, c_k\}\}$. Let $\mathsf{Supp}^k(Q, D, \bar{a})$ be the restriction of $\mathsf{Supp}(Q, D, \bar{a})$ to $\mathsf{V}^k(D)$, i.e., $\mathsf{Supp}(Q, D, \bar{a}) \cap \mathsf{V}^k(D)$. Then we define

$$\mu^k(Q, D, \bar{a}) \;=\; \frac{|\mathsf{Supp}^k(Q, D, \bar{a})|}{|\mathsf{V}^k(D)|}$$

as the proportion of valuations from $\mathsf{V}^k(D)$ that belong to $\mathsf{Supp}(Q, D, \bar{a})$. Finally, we look at the asymptotic behavior of this sequence:

$$\mu(Q, D, \bar{a}) = \lim_{k\to\infty} \mu^k(Q, D, \bar{a})\,.$$

This definition assumes some particular enumeration of the set Const. However, it is easy to see that the limit value $\mu(Q, D, \bar{a})$ is independent of a particular enumeration for C-generic queries: once the set $\{c_1, \ldots, c_k\}$ contains C and $\mathsf{Const}(D)$, the value $\mu^k(Q, D, \bar{a})$ does not depend on the remaining elements of this set, just their number. Hence, we always assume, without any loss of generality, some fixed enumeration of Const.

For Boolean queries, we write $\mu(Q, D)$ instead of the more formal $\mu(Q, D, ())$ (recall that the empty tuple $()$ denotes *true*), and likewise for μ^k. That is, $\mu^k(Q, D) = |\{v \in \mathsf{V}^k(D) \mid v(D) \models Q\}| \,/\, |\mathsf{V}^k(D)|$ and $\mu(Q, D)$ is the limit of this sequence.

Definition 4 (Almost certainly true/false answers). *Given a query Q, a database D, and a tuple $\bar{a}$, we say that $\bar{a}$ is an* almost certainly true *answer to Q on D if $\mu(Q, D, \bar{a}) = 1$, and* almost certainly false *answer if $\mu(Q, D, \bar{a}) = 0$.*

It is immediate from the definitions that every certain answer $\bar{a} \in \Box(Q, D)$ is almost certainly true, and every answer that is not possible is almost certainly false. We next prove a dichotomy result for arbitrary generic queries, stating that:

- every tuple is either an almost certainly true answer or an almost certainly false answer; and
- almost certainly true answers are exactly those returned by the naïve evaluation of Q.

Theorem 1 (0–1 law). *If Q is a generic query, then for every database D and every tuple $\bar{a}$ over the active domain, $\mu(Q, D, \bar{a})$ is either 0 or 1. Furthermore, $\mu(Q, D, \bar{a}) = 1$ iff $\bar{a} \in Q^{\text{naïve}}(D)$.*

Before proving this, we state several corollaries. Since $\mu(Q, D, \bar{a}) = 1$ for certain answers, we obtain a generalization of a result that was explicitly stated for FO queries before [32]:

Corollary 1. $\Box(Q, D) \subseteq Q^{\text{naïve}}(D)$ *for every generic query.* □

It is also much easier to find almost certainly true answers than certain answers:

Corollary 2. *Checking whether $\bar{a}$ is an almost certainly true answer to Q on D has the same data complexity as the evaluation Q on D.* □

Notice that in the study of 0–1 laws, the situation is similar: checking if an FO sentence is valid is undecidable, while checking if it is valid almost everywhere is decidable in PSPACE, see [23].

Finally, for some queries, certain answers and almost certainly true answers coincide. Consider, for instance, a fragment of FO, denoted $\text{Pos}^{\forall\text{G}}$ (for positive FO with universal guards), that contains atomic formulae and is closed under conjunction, disjunction, quantification (both existential and universal) and the following rule: if φ is a formula in the fragment, then so is $\forall \bar{x}\ \alpha(\bar{x}) \rightarrow \varphi$, where α is an atom and $\bar{x}$ is a tuple of distinct variables [14]. For $\text{Pos}^{\forall\text{G}}$ queries, naïve evaluation is known to produce certain answers [22]. Thus, we have:

Corollary 3. *For $\text{Pos}^{\forall\text{G}}$ queries, certain answers and almost certainly true answers are the same.* □

Proof idea of Theorem 1. First we show that by a simple transformation of the query (essentially asking if a relation containing tuple $\bar{a}$ belongs to $Q(D)$) we can reduce the problem to Boolean queries; thus in this proof and elsewhere it will suffice to consider $\mu(Q, D)$ for Boolean queries.

The main idea is to introduce a different measure $\mu^k_{\mathrm{bij}(C)}(Q,D)$ which is the proportion of C-bijective valuations v with $\mathrm{range}(v) \subseteq \{c_1,\ldots,c_k\}$ such that $v(D) \models Q$; here we assume Q to be C-generic. We then prove that $\mu(Q,D) = \lim_{k\to\infty} \mu^k_{\mathrm{bij}(C)}(Q,D)$. To show this, we calculate an upper bound on the proportion of non-C-bijective valuations, by analyzing conditions that make valuations non-bijective, and show that it approaches 0 for large k.

Now assume that Q naïvely evaluates to true. By Proposition 1 it means that $v(D) \models Q$ for all C-bijective valuations, and thus $\mu(Q,D) = \lim_{k\to\infty} \mu^k_{\mathrm{bij}(C)}(Q,D) = 1$. If Q naïvely evaluates to false, we apply this argument to the negation of Q, which remains C-generic, and get $\mu(Q,D) = 0$. □

Remark: connections with 0–1 laws in logic. Even though there are obvious parallels with the study of 0–1 laws in logic, the nature of Theorem 1 is quite different from its logic counterparts, which explains why it holds even for queries in logics that do not posses the 0–1 law. When one proves 0–1 laws in logic, with increasing k we have an increasing number of isomorphism types of structures on k elements. To prove the 0–1 law, one has to analyze their behavior with respect to the satisfaction of some logical sentences, which is usually done by a games argument. In our setting, to the contrary, at some point the number of isomorphism types stabilizes. This does not mean that the sequence μ^k stabilizes (in fact it usually does not) but it means that the combinatorial arguments involved in our proofs are rather different.

Remark. We also note that computing each individual value $\mu^k(Q,D)$ for $k \in \mathbb{N}$ can be cast as the problem of query evaluation on a probabilistic database. Indeed, to each tuple $\bar{c}$ over $\{c_1,\ldots,c_k\}$ of arity of a relation R in a database D, assign the probability that is the proportion of valuations $v \in \mathsf{V}^k(D)$ such that $v(\bar{a}) = \bar{c}$ for some $\bar{a} \in R$; then $\mu^k(Q,D)$ is the result of evaluating Q over such a probabilistic database, whose size is exponential in k. However, the field of probabilistic databases [40] concentrates on very different problems, mainly analyzing the complexity of query evaluation (which is often high, even for fairly simple queries). This is very different from our focus. We are not interested in computing $\mu^k(Q,D)$, nor its complexity; rather, we analyze the asymptotic behavior of the sequence of numbers $\mu^k(Q,D)$.

3.3 An alternative measure

There is an alternative way of defining the measure of certainty: instead of counting the number of valuations that make a query true on a given database, we can count the number of complete databases that make the query true. That is, we can look not at $|\{v \in \mathsf{V}^k(D) \mid v(\bar{a}) \in Q(v(D))\}|$, but rather at $|\{v(D) \mid v \in \mathsf{V}^k(D) \text{ and } v(\bar{a}) \in Q(v(D))\}|$. These cardinalities could be different because we may well have $v_1(D) = v_2(D)$ for different v_1 and v_2. For instance if D contains a binary relation with tuples $(1,\bot)$ and $(1,\bot')$, and v_1 is an arbitrary valuation, then for valuation v_2 defined by $v_2(\bot) = v_1(\bot')$ and $v_2(\bot') = v_1(\bot)$ we have $v_1(D) = v_2(D)$.

Thus we can alternatively define

$$\mathfrak{m}^k(Q,D,\bar{a}) \;=\; \frac{|\{v(D) \mid v \in \mathsf{Supp}^k(Q,D,\bar{a})\}|}{|\{v(D) \mid v \in \mathsf{V}^k(D)\}|} \qquad (1)$$

and let $\mathfrak{m}(Q,D,\bar{a})$ be $\lim_{k\to\infty} \mathfrak{m}^k(Q,D,\bar{a})$. When Q is a Boolean query, we can view this measure in yet another way by using the notion of the semantics of an incomplete database. Recall that $[\![D]\!] = \{v(D) \mid v \in \mathsf{V}(D)\}$. Let $[\![D]\!]^k$ be the restriction of $[\![D]\!]$ to databases D' with $\mathrm{adom}(D') \subseteq \{c_1,\ldots,c_k\}$. If k is large enough so that this set contains all constants in D, we have

$$\mathfrak{m}^k(Q,D) \;=\; \frac{|\{D' \in [\![D]\!]^k \mid D' \models Q\}|}{|[\![D]\!]^k|} \qquad (2)$$

As mentioned earlier, similar considerations enter the picture when one deals with 0–1 laws in logic, and measures the number of isomorphism types of structures that make a sentence true. However, while the definitions differed slightly, in the end they defined the same measure. The same is true here:

Theorem 2. *For every generic query Q, database D, and tuple $\bar{a}$, we have $\mu(Q,D,\bar{a}) = \mathfrak{m}(Q,D,\bar{a})$.*

Proof sketch. As in the proof of Theorem 1, the idea is to reduce everything to the case of bijective valuations. Also as there, it suffices to consider Boolean queries. Let Q be C-generic. Define $\mathsf{Supp}^k_{\mathrm{bij}(C)}(Q,D)$ as the set of C-bijective valuations with the range contained in $\{c_1,\ldots,c_k\}$ such that $v(D) \models Q$. Let

$$\mathfrak{m}^k_{\mathrm{bij}(C)}(Q,D) \;=\; \frac{|\{v(D) \mid v \in \mathsf{Supp}^k_{\mathrm{bij}(C)}(Q,D)\}|}{|\{v(D) \mid v \in \mathsf{V}^k(D)\}|}\,.$$

Then the result follows from the observation made in the proof of Theorem 1 that $\mu(Q,D) = \lim_{k\to\infty} \mu^k_{\mathrm{bij}(C)}(Q,D)$ and two equalities below:

$$\begin{aligned} \lim_{k\to\infty} \mathfrak{m}^k_{\mathrm{bij}(C)}(Q,D) &= \lim_{k\to\infty} \mathfrak{m}^k(Q,D) \\ \lim_{k\to\infty} \mathfrak{m}^k_{\mathrm{bij}(C)}(Q,D) &= \lim_{k\to\infty} \mu^k_{\mathrm{bij}(C)}(Q,D)\,. \end{aligned}$$

To prove the first equality, it will suffice to show that the ratio of the number of different $v(D)$ for non-C-bijective valuations and the number of all $v(D)$, when v ranges over $\mathsf{V}^k(D)$, goes to 0 for large k. For this we prove an upper bound for the former and a lower bound for the latter, as $(m^2+mc)\cdot k^{m-1}$ and $\binom{k-c}{m}$, where m is the number of nulls in D, and c is the number of constants in D plus $|C|$. Then we show that their ratio approaches 0 for large k.

To prove the second equality, we analyze cases when $v_1(D) = v_2(D)$ for C-bijective valuations, and show that it only happens if one is obtained from the other by composing with an automorphism on nulls of D; this allows us to relate $\mathfrak{m}^k_{\mathrm{bij}(C)}(Q,D)$ and $\mu^k_{\mathrm{bij}(C)}(Q,D)$ and prove that they have the same limits. □

3.4 Open world semantics

We now briefly address the open world semantics. Under this semantics, an incomplete database D represents the set $[\![D]\!]_{\mathrm{OWA}} = \{v(D) \cup D' \mid v \text{ is a valuation and } D' \text{ is finite and complete}\}$. In other words, the database is open to new facts: once values for nulls have been obtained by means of a valuation v, an arbitrary finite set of complete tuples can be added. A Boolean query Q is true with certainty in D under OWA if Q is true in every database from $[\![D]\!]_{\mathrm{OWA}}$ [1, 27].

The measure $\mathfrak{m}$ can easily be adapted to the OWA case: one simply takes the definition (2) and replaces the usual semantics $[\![\,]\!]$

with $[\![\ ]\!]_{\text{OWA}}$ to obtain the value $\text{owa-}\mathfrak{m}^k(Q,D)$; then

$$\text{owa-}\mathfrak{m}(Q,D) \;=\; \lim_{k\to\infty} \text{owa-}\mathfrak{m}^k(Q,D)\,,$$

whenever the limit exists.

However, for the open world semantics, we lose the connection between naïve evaluation and the behavior of such a measure. This is not surprising of course, given the complete freedom to add arbitrary databases D' to those obtained by assigning values to nulls. We now state this formally.

Proposition 2. *There exist first-order queries Q_1 and Q_2 such that*

- $\text{owa-}\mathfrak{m}(Q_1,D) = 1$ *but* $Q_1^{\text{naïve}}(D) = \text{false}$*; and*
- $\text{owa-}\mathfrak{m}(Q_2,D) = 0$ *but* $Q_2^{\text{naïve}}(D) = \text{true}$.

Proof. Consider a database D with a single unary relation U which is empty. Let Q_1 be $\neg\exists x\ U(x)$. Then $Q_1^{\text{naïve}}(D) = \mathit{true}$ while $\text{owa-}\mathfrak{m}^k(Q_1,D) = 2^{-k}$ and thus $\text{owa-}\mathfrak{m}(Q_1,D) = 0$. For $Q_2 = \exists x\ U(x)$ (i.e., $Q_2 = \neg Q_1$) we then have $Q_2^{\text{naïve}}(D) = \mathit{false}$ and $\text{owa-}\mathfrak{m}(Q_2,D) = 1$. □

4 MEASURING CERTAINTY UNDER CONSTRAINTS

In real life databases must satisfy integrity constraints, most commonly keys and foreign keys, which are special cases of functional dependencies and inclusion constraints (which in turn are special cases of equality- and tuple-generating dependencies). A set of constraints Σ can be viewed as a Boolean query, returning *true* if constraints are satisfied and *false* if they are not. We always assume this query is generic. This is true for all real-life constraints, certainly for all those mentioned above, since they are definable by logical formulae. The question that we address is: how can we measure certainty under constraints?

In general, we know much less about certain answers to queries over incomplete databases under constraints. The notion of certainty of a Boolean query Q under constraints is that Q must be true in every database satisfying constraints [12, 13, 25] (for non-Boolean queries one requires that $Q(\bar{a})$ be true in every such database). This is the same as requiring that the implication $\Sigma \to Q$ be true over all databases. However, even for well-behaved queries Q and simple constraints Σ, such as keys and inclusion constraints, queries of the form $\neg\Sigma \vee Q$ are usually not in classes that exhibit good behavior with respect to finding certain answers. This explains why most results on query answering under constraints over incomplete databases are actually negative [12, 13].

To extend the measure $\mu(Q,D)$ to deal with constraints, there are two approaches.

- *Measuring certainty of* $\Sigma \to Q$: in other words, we simply consider $\mu(\Sigma \to Q, D)$. While using implication is standard, this approach is actually not very interesting as it gives us little new information: $\mu(\Sigma \to Q, D)$ is either 1, or coincides with $\mu(Q,D)$, and it is easy to check which case applies. This will be shown in Proposition 3.
- *Measuring conditional probability*: once we think of the measure μ in probabilistic terms, this is a much more natural approach. We want to find the conditional probability of Q, given that Σ holds. In other words, choose a random assignment of values to nulls; what is the probability that Q is true under the assumption that Σ is true?

To see that we get a much richer picture with the second approach, consider a database D with relations $R = \{(2,1),(\bot,\bot)\}$ and $U = \{1,2,3\}$, and let Σ contain the inclusion constraint $\forall x,y\ \big(R(x,y) \to U(x)\big)$. Assume that query Q simply returns R. The constraint tells us that $\bot$ can only be assigned values 1, 2, or 3. If we take $\bar{a} = (1,\bot)$, then conditional probability of $Q(\bar{a})$ being true under Σ is $\frac{1}{3}$, since only the valuation $\bot \mapsto 1$ makes $Q(\bar{a})$ true. On the other hand, for $\bar{b} = (2,\bot)$, assignments $\bot \mapsto 1$ and $\bot \mapsto 2$ make $Q(\bar{b})$ true, while $\bot \mapsto 3$ does not, and thus the conditional probability of $Q(\bar{b})$ being true is $\frac{2}{3}$.

We now investigate both approaches, starting with the easy case of implication $\Sigma \to Q$, and then moving on to the conditional probability approach.

4.1 Measuring implication

Assume, without any loss of generality, that Q is Boolean, using the same argument as in the previous section. By Theorem 1, we know that $\mu(\Sigma,D)$ is 0 or 1 for generic Σ. These two cases fully determine the behavior of $\mu(\Sigma \to Q, D)$.

Proposition 3. *Assume that both Σ and Q are generic. Let D be a database.*

- *If $\mu(\Sigma,D) = 0$, then $\mu(\Sigma \to Q, D) = 1$.*
- *If $\mu(\Sigma,D) = 1$, then $\mu(\Sigma \to Q, D) = \mu(Q,D)$.* □

Proof sketch. First we note that $\mu^k(\Sigma \to Q, D)$ is at least $|\text{Supp}^k(\neg\Sigma, D)| \,/\, |\text{V}^k(D)| = 1 - \mu^k(\Sigma, D)$ and thus $\mu(\Sigma \to Q, D) = 1$ if $\mu(\Sigma,D) = 0$.

Now assume $\mu(\Sigma,D) = 1$. If $\mu(Q,D) = 1$, then $\mu^k(\neg\Sigma \vee Q, D) \geq \mu^k(Q,D)$ and thus $\mu(\Sigma \to Q, D) = 1$ too. If $\mu(Q,D) = 0$, then for every $\varepsilon > 0$, there exists k_0 such that for each $k > k_0$ we have both $\frac{|\text{Supp}^k(\neg\Sigma,D)|}{|\text{V}^k(D)|} < \frac{\varepsilon}{2}$ and $\frac{|\text{Supp}^k(Q,D))|}{|\text{V}^k(D)|} < \frac{\varepsilon}{2}$ which implies that $\mu^k(\Sigma \to Q, D) \leq \Big(|\text{Supp}^k(\neg\Sigma,D)| + |\text{Supp}^k(Q,D)|\Big) \,/\, |\text{V}^k(D)| < \varepsilon$, and thus $\mu(\Sigma \to Q, D) = 0$. Since $\mu(Q,D)$ can only be 0 or 1, this proves $\mu(\Sigma \to Q, D) = \mu(Q,D)$. □

4.2 Conditional probabilities and convergence

We now define the conditional probability $\mu(Q|\Sigma, D, \bar{a})$ of $\bar{a}$ being an answer to Q, given that Σ holds. For this, we first define

$$\begin{aligned}\mu^k(Q|\Sigma,D,\bar{a}) \;&=\; \frac{\mu^k(\Sigma \wedge Q, D, \bar{a})}{\mu^k(\Sigma,D)}\\ &=\; \frac{|\text{Supp}^k(\Sigma \wedge Q, D, \bar{a})|}{|\text{Supp}^k(\Sigma,D)|}\end{aligned}$$

that is, the probability that a randomly chosen valuation v with $\text{range}(v) \subseteq \{c_1,\ldots,c_k\}$ such that $v(D) \models \Sigma$ also satisfies $v(\bar{a}) \in Q(v(D))$. In the case of Boolean queries Q, this is the probability that a randomly chosen valuation witnessing Σ also witnesses Q.

We then define

$$\mu(Q|\Sigma,D,\bar{a}) \;=\; \lim_{k\to\infty} \mu^k(Q|\Sigma,D,\bar{a})$$

if such a limit exists.

Of course it is possible that the denominator $|\text{Supp}^k(\Sigma, D)|$ is zero, if $\text{Supp}^k(\Sigma, D) = \emptyset$. We could restrict the definition of $\mu^k(Q|\Sigma, D, \bar{a})$ and let it be defined only if Σ is satisfiable in D, i.e., $v(D) \models \Sigma$ for at least one valuation v; this ensures that $|\text{Supp}^k(\Sigma, D)| > 0$ for all sufficiently large k. For simplicity of presentation, and to avoid carrying this assumption throughout, we adopt the convention that $\mu^k(Q|\Sigma, D, \bar{a}) = 0$ if $\text{Supp}^k(\Sigma, D) = \emptyset$.

The same argument we used in the previous section shows that $\mu(Q|\Sigma, D, \bar{a})$ does not depend on a particular enumeration of Const; in fact, for C-generic Q and Σ, once k is big enough so that both C and $\text{Const}(D)$ are in $\{c_1, \ldots, c_k\}$, the value of $\mu^k(Q|\Sigma, D, \bar{a})$ does not depend on a particular enumeration.

As we saw, the limit, even if it exists, need not be zero or one: our earlier example showed that it could be 1/3 or 2/3. As we saw in Section 2, the next best thing in the case of failure of the 0–1 law is convergence, i.e., the existence of the limit. This is what we prove now.

Theorem 3 (Convergence). *If both Σ and Q are generic, then for every database D and every tuple $\bar{a}$ over the active domain, $\mu(Q|\Sigma, D, \bar{a})$ exists and is a rational number in $[0, 1]$.*

Proof sketch. We prove a slightly more general result. Let Q_1 and Q_2 be Boolean queries. We assume that Q_2 is satisfiable in D, i.e., $v(D) \models Q_2$ for some v. Define $\mu^k(Q_1/Q_2, D)$ as $\mu^k(Q_1, D)/\mu^k(Q_2, D) = |\text{Supp}^k(Q_1, D)|/|\text{Supp}^k(Q_2, D)|$. We then show that either $\lim_{k\to\infty} \mu^k(Q_1/Q_2, D)$ exists and is a number in $\mathbb{Q}$, or $\mu^k(Q_1/Q_2, D)$ grows arbitrarily large with k. Since $\mu^k(Q|\Sigma, D)$ is $\mu^k(Q \wedge \Sigma/\Sigma, D)$, these numbers can never exceed 1, and thus the latter case does not arise, which gives us the result.

To prove the above result, fix an arbitrary C-generic Boolean query q and a database D with m nulls, and let $A = C \cup \text{Const}(D)$. We assume that k is large enough so that $A \subset \{c_1, \ldots, c_k\}$ and $k > |A| + m$. Then we show how to express the quantity $|\text{Supp}^k(q, D)|$ as a polynomial in $\mathbb{Q}[k]$. The result then follows since the limit of the sequence $|\text{Supp}^k(Q_1, D)|/|\text{Supp}^k(Q_2, D)|$ is the ratio of the leading coefficients of the polynomials expressing these cardinalities if their degrees are the same (and 0 or ∞ when their degrees differ; recall that in this statement Q_1 and Q_2 are arbitrary queries; when $Q_2 = Q_1 \wedge \Sigma$, the latter option is excluded).

To express $|\text{Supp}^k(q, D)|$ as a polynomial in $\mathbb{Q}[k]$, we consider a partition ρ on the nulls of D and look at all valuations v that output the same value for each null in the same block. We then define an equivalence relation on such valuations so that if v and v' are equivalent valuations, then $v(D)$ and $v'(D)$ agree on q, i.e., $v(D) \models q$ iff $v'(D) \models q$. Furthermore, this equivalence relation is defined in such a way that the number of equivalence classes does not depend on k. Then, for each equivalence class, we show how to express the number of valuations in it as a polynomial in k. Since the number of permutations on the nulls and the number of equivalence classes do not depend on k, this means that $|\text{Supp}^k(q, D)|$ can be expressed as a polynomial in k. □

In the study of 0–1 law and convergence in logic, one often finds that asymptotic behaviors of logical properties are described by rationals of a particular kind, e.g., $a/2^b$ [17, 34]. In our case, it turns out, there is no restriction on the numbers that can appear as $\mu(Q|\Sigma, D)$, even for simple constraints and queries.

Proposition 4. *For every $s \in \mathbb{Q} \cap [0, 1]$, there is a database D, an inclusion constraint Σ, and a Boolean conjunctive query Q such that $\mu(Q|\Sigma, D) = s$.*

Proof sketch. Let $s = p/r$, with $0 < p \leq r$. Consider a database D with two binary relations R and S and a unary relation U, where $R = \{(1, 1), (2, 2), \ldots, (p-1, p-1), (\bot, p)\}$, relation S contains a single tuple $(\bot, \bot)$, and U contains all numbers from 1 to r. Let Σ contain the single inclusion constraint $\pi_1(R) \subseteq U$ (i.e., the first column of R is contained in U), and let Q be $\exists x, y\ R(x, y) \wedge S(x, y)$. There are only r valuations v such that $v(D) \models \Sigma$, those that send $\bot$ to $1, \ldots, r$. Then one shows that $v(D) \models Q$ iff $v(\bot) \leq p$, which proves $\mu(Q|\Sigma, D) = p/r$. □

4.3 Complexity of $\mu(Q|\Sigma, D)$

We have seen that without constraints, computing $\mu(Q, D)$ is easy: we just evaluate Q naïvely on D. But even simple constraints break this connection. As an example, consider a database D with four unary relations: $R = \{\bot\}$, $S = \{\bot'\}$, $U = \{\bot\}$, and $V = \{1\}$. Let Σ contain unary inclusion constraints $R \subseteq V$ and $S \subseteq V$, and let Q be the first-order Boolean query $\forall x\ U(x) \rightarrow (R(x) \wedge \neg S(x))$. Then both $Q^{\text{naïve}}(D)$ and $(\Sigma \rightarrow Q)^{\text{naïve}}(D)$ evaluate to *true*, but $\mu(Q|\Sigma, D) = 0$.

We thus investigate the complexity of computing $\mu(Q|\Sigma, D)$. First, we need to explain what we actually mean by computing $\mu(Q|\Sigma, D)$ and what upper and lower complexity bounds are appropriate for it. Since, by Theorem 3, $\mu(Q|\Sigma, D)$ can be an arbitrary rational number in $[0, 1]$, we can represent it as a pair (p, r) with $p, r \in \mathbb{N}$ so that $\mu(Q|\Sigma, D) = p/r$. Thus, computing $\mu(Q|\Sigma, D)$ is a problem in a *function* class (rather than a complexity class capturing decision problems). Furthermore, it amounts to computing two numbers. In complexity theory, we have classes for decision problems, or problems of computing a single number in $\mathbb{N}$, e.g., #P. Hence, providing precise complexity characterization that leaves no gaps requires a bit of care.

The standard approach in this case is to show that the problem lies in the class $\text{FP}^{\#\text{P}}$ of functions computable in polynomial time with an access to a #P oracle, and that there are cases when $\mu(Q|\Sigma, D) = p/r$ such that r can be computed in polynomial time while computing p is #P-hard. This indeed leaves no gap since it is known that a problem is $\text{FP}^{\#\text{P}}$-hard iff it is #P-hard [20].

The upper bound can be obtained from the construction presented in the proof of Theorem 3. It showed that the limit is the ratio of the leading coefficients of two polynomials, and these coefficients are obtained by counting equivalence classes of valuations. This gives us

Proposition 5. *Given generic constraints Σ and a generic query Q of polynomial data complexity, computing $\mu(Q|\Sigma, D)$ is in $\text{FP}^{\#\text{P}}$.* □

To show that we cannot do better, we next prove the #P-hardness lower bound for finding the numerator of $\mu(Q|\Sigma, D)$. In fact we need a stronger result. Note that by default, $\mu(Q|\Sigma, D) = 0$ if Σ is not satisfiable in D (i.e., no $v(D)$ satisfies Σ). Checking whether Σ is not satisfiable in D can be intractable even for keys and foreign keys, as one can easily encode the complement of the homomorphism problem. Obtaining computational hardness simply by recourse to satisfiability is not fair however as it mixes two different

problems; we would rather prove a stronger hardness result only for constraints whose satisfiability is tractable. We show such a result for *unary* keys and foreign keys. Note that we interpret these constraints in the same way as they are interpreted in RDBMSs, i.e., attributes declared as keys cannot be nulls.

Proposition 6. *If Σ is a set of unary keys and foreign keys over a fixed schema, and D is an incomplete database, then satisfiability of Σ in D can be solved in polynomial time in data complexity. Moreover, there exist a single unary foreign key Σ and a first-order*[1] *query Q such that $\mu(Q|\Sigma, D) = \frac{p(D)}{r(D)}$ for every database D, where $r(D)$ can be computed in linear time but computing $p(D)$ is #P-complete.* □

4.4 0–1 law with constraints

We have seen that the value of the measure $\mu(Q|\Sigma, D)$ of the certainty of answer to Q under constraints Σ could be any rational number in $[0, 1]$, even for conjunctive queries. In some cases, however, we can recover the 0–1 law. These are the cases of almost certainly true constraints, and of constraints that only include functional dependencies. In both of these cases we also have polynomial-time algorithms for computing $\mu(Q|\Sigma, D)$.

0–1 law for almost certainly true constraints. If the set of constraints Σ is generic, then by Theorem 1, $\mu(\Sigma, D)$ is 0 or 1. It turns out that arbitrary rational numbers can only be seen as values of $\mu(Q|\Sigma, D)$ if $\mu(\Sigma, D) = 0$; otherwise we recover the 0–1 law. More precisely, if constraints of Σ hold in D when evaluated naïvely, then they do not affect the value of $\mu(Q|\Sigma, D, \bar{a})$.

Theorem 4. *If Σ and Q are generic, and D is a database such that $\Sigma^{\text{naïve}}(D) = \text{true}$, then for every tuple $\bar{a}$ over the active domain of D, we have $\mu(Q|\Sigma, D, \bar{a}) = \mu(Q, D, \bar{a})$.* □

Proof sketch. Consider Boolean queries Q; by Theorem 1, $\mu(\Sigma, D) = 1$ and $\mu(Q, D) \in \{0, 1\}$. If $\mu(Q, D) = 0$, then, for every small ε and all sufficiently large k we have $\mu^k(Q, D) < \varepsilon/2$ and $\mu^k(\Sigma, D) > 1 - \varepsilon/2$. Thus $\mu^k(Q|\Sigma, D) \leq (\varepsilon/2)/(1 - \varepsilon/2) < \varepsilon$ and hence $\mu(Q|\Sigma, D) = 0$. If $\mu(Q, D) = 1$, then again for each small ε and all sufficiently large k we have $\mu^k(Q, D) \geq 1 - \varepsilon/2$ and $\mu^k(\Sigma, D) > 1 - \varepsilon/2$, from which $\mu^k(Q|\Sigma, D) > 1 - \varepsilon$ can be similarly derived; thus $\mu(Q|\Sigma, D) = 1$. □

0–1 law for functional dependencies. To get arbitrary rational numbers as values of $\mu(Q|\Sigma, D)$, we needed inclusion constraints. If we use only functional dependencies (FDs), the 0–1 law is recovered, and we can effectively decide whether the value is 0 or 1. The procedure is this: chase the incomplete database with constraints, and then apply the measure μ to the query and the result of the chase.

We now briefly recall the chase procedure for FDs (cf. [1, 25]). We can assume without loss of generality that all FDs in the set Σ are of the form $X \to A$, where X is a set of attributes, and A is an attribute. Then chasing a database D with constraints Σ refers to the following procedure. At each step, it chooses a violation of an FD from Σ, i.e., two tuples t_1, t_2 in a relation R of D, and an FD $X \to A$ over R, such that $\pi_X(t_1) = \pi_X(t_2)$, but $\pi_A(t_1) \neq \pi_A(t_2)$ and does the following:

- if one of $\pi_A(t_1), \pi_A(t_2)$ is an element of $\text{Null}(D)$ and the other is a constant, then the null is replaced by the constant everywhere in D;
- if both $\pi_A(t_1), \pi_A(t_2)$ are elements of $\text{Null}(D)$, then one is replaced by the other everywhere in D;
- if both $\pi_A(t_1), \pi_A(t_2)$ are elements of $\text{Const}(D)$, then the procedure fails.

These steps are repeated as long as they can be; if the procedure did not fail and no step applies anymore, then it terminates successfully, and its output is denoted by $\text{chase}_\Sigma(D)$. While the chase procedure is non-deterministic, every sequence of chase steps results in the same instance, up to renaming of nulls, cf. [1, 3]; hence we can call it $\text{chase}_\Sigma(D)$ without ambiguity. Its running time is polynomial in the size of D.

Among its many uses, the chase is known to help compute certain answers to conjunctive queries under constraints [13, 25]. In fact, if Σ is a set of FDs and chase terminates successfully on a database D, then a tuple $\bar{a}$ of constants is a certain answer to a conjunctive query Q on D under Σ iff $\bar{a} \in Q^{\text{naïve}}(\text{chase}_\Sigma(D))$, cf. [25]. Note that this statement only makes sense for constant tuples, since chase can rename nulls.

We show that the same statement is true not only for conjunctive queries, but for *all* generic queries, if we replace certain answers by almost certainly true answers.

Theorem 5. *Let Σ be a set of FDs, Q a generic query and D a database. If $\bar{a}$ is a tuple of constants from D, then*

$$\mu(Q|\Sigma, D, \bar{a}) \;=\; \mu(Q, \text{chase}_\Sigma(D), \bar{a}).$$

We assumed, by convention, that $\mu(Q, \text{chase}_\Sigma(D), \bar{a})$ is set to 0 if chasing D with Σ fails.

Theorems 1 and 5 give us the following.

Corollary 4. *Let Σ be a set of FDs, Q a generic query, D a database, and $\bar{a}$ a tuple of constants from D. Assume that chasing D with Σ terminates successfully. Then*

$$\mu(Q|\Sigma, D, \bar{a}) = \begin{cases} 1 & \text{if } \bar{a} \in Q^{\text{naïve}}(\text{chase}_\Sigma(D)), \\ 0 & \text{otherwise}. \end{cases}$$

Proof idea of Theorem 5. Applying the chase can collapse some valuations v, v' on D to the same valuation on $\text{chase}_\Sigma(D)$, and thus we first need to analyze when this happens, and in particular how many valuations on D can collapse to the same one on the result of the chase. Next we assume that $\mu(Q, \text{chase}_\Sigma(D)) = 1$ with the goal of showing that $\mu(Q|\Sigma, D) = 1$. For this, we under- and over-approximate the numerator and denominator of the expression for $\mu^k(Q|\Sigma, D)$ in terms of valuations from $\mathsf{V}^k(\text{chase}_\Sigma(D))$, in particular using the above analysis on how they interact with the homomorphism that sends D to $\text{chase}_\Sigma(D)$. Using such approximations, we then prove that the sequence $\mu^k(Q|\Sigma, D)$ converges to 1. The case when $\mu(Q, \text{chase}_\Sigma(D)) = 0$ is handled by applying the same argument to the negation of Q. □

5 COMPARING QUERY ANSWERS

So far we defined measures of certainty of query answers as probabilities that a randomly chosen valuation witnesses that a tuple is in the answer. Now we shift to qualitative rather than quantitative

[1] It was observed by M. Console that by modifying the reduction, the query can be taken to be conjunctive.

comparisons, and declare one answer to be *better* than another if it has more support. That is, given a database D, a k-ary query Q, and k-tuples $\bar{a}, \bar{b}$ over the active domain, we define

$$\bar{a} \trianglelefteq_{Q,D} \bar{b} \quad \Leftrightarrow \quad \mathsf{Supp}(Q, D, \bar{a}) \subseteq \mathsf{Supp}(Q, D, \bar{b})$$

$$\bar{a} \triangleleft_{Q,D} \bar{b} \quad \Leftrightarrow \quad \mathsf{Supp}(Q, D, \bar{a}) \subset \mathsf{Supp}(Q, D, \bar{b}) .$$

Already in the introduction, we saw that these notions apply even if we have tuples for which $\mu(Q, D, \bar{a}) = \mu(Q, D, \bar{b}) = 1$. In such a case, we may still have $\bar{a} \triangleleft_{Q,D} \bar{b}$, indicating that $\bar{b}$ should be a preferred answer compared to $\bar{a}$, even if both are almost certainly true answers.

Since $\bar{a} \triangleleft_{Q,R} \bar{b}$ indicates that $\bar{b}$ is a better answer, it is natural to define the set of *best* answers as those for which there is no better one:

$$\mathsf{Best}(Q, D) = \{\bar{a} \mid \neg\exists \bar{b} : \bar{a} \triangleleft_{Q,D} \bar{b}\} .$$

The notion of best answers $\mathsf{Best}(Q, D)$ is closely related to certain answers $\Box(Q, D)$, but has some advantages. Certain answers $\Box(Q, D)$ could be empty. However, if D is not empty, then $\mathsf{Best}(Q, D) \neq \emptyset$ by definition. At the same time, if $\Box(Q, D) \neq \emptyset$, then $\mathsf{Best}(Q, D) = \Box(Q, D)$. Thus, $\mathsf{Best}(Q, D)$ asks for the best answers to Q one can get; if certain answers exist, they are the best, but otherwise we can still talk about best answers even if there are no certain answers.

To give an example, consider a database D with relations $R = \{(1, \bot_1), (2, \bot_2)\}$ and $S = \{(1, \bot_2), (\bot_3, \bot_1)\}$, and query $Q(x, y) = R(x, y) \wedge \neg S(x, y)$ computing their difference. It is easy to see that $\Box(Q, D) = \emptyset$. For $\mathsf{Best}(Q, D)$, there are two natural candidate tuples: $\bar{a} = (1, \bot_1)$ and $\bar{b} = (2, \bot_2)$. Note that $v(\bar{a}) \in Q(v(D))$ iff $v(\bot_1) \neq v(\bot_2)$ and $v(\bot_3) \neq 1$, while $v(\bar{b}) \in Q(v(D))$ iff $v(\bot_1) \neq v(\bot_2)$ or $v(\bot_3) \neq 2$. Thus, $\bar{a} \triangleleft_{Q,D} \bar{b}$ and $\mathsf{Best}(Q, D) = \{\bar{b}\}$. In this case, even if certain answers are empty, we could find the best answer to provide the user with.

In this section we study the complexity of comparing answers and finding best answers. For this, we need to cast them as decision problems. For relations $\theta \in \{\trianglelefteq, \triangleleft\}$, the natural decision problem is

Problem:	θ-Comparison
Input:	A query Q, a database D, tuples $\bar{a}, \bar{b}$
Question:	Is $\bar{a}\ \theta_{Q,D}\ \bar{b}$?

We turn finding best answers into a decision problem by asking whether the best answer belongs to a specified family of sets (candidate answers):

Problem:	BestAnswer
Input:	A query Q, a database D, a family $\mathcal{X}$ of sets of tuples
Question:	Is $\mathsf{Best}(Q, D) \in \mathcal{X}$?

Other formulations as a decision problem are possible too, but this is enough to understand where the problem lies in terms of its complexity. We look at *data complexity* of these problems, when query Q is fixed. We establish it for first-order queries, and some sublanguages. As usual, we say that data complexity is complete for a complexity class for a given language, if for each query in the language, data complexity is in that class, and for some, it is hard for the class. Recall that testing whether a tuple belongs to $\Box(Q, D)$ is coNP-complete in data complexity for FO queries, and tractable for unions of conjunctive queries [2].

The complexity of the problems we study falls within the second level of the polynomial hierarchy, but low in that level, in $\Sigma_2^p \cap \Pi_2^p$. This holds not only for first-order queries, but also for queries whose evaluation is tractable in data complexity. We start with comparisons of tuples.

Theorem 6. *For first-order queries, $\trianglelefteq$-Comparison is* coNP*-complete and $\triangleleft$-Comparison is* DP*-complete in data complexity. In fact, membership in* coNP *and* DP *holds for arbitrary queries whose data complexity is polynomial.*

Proof sketch. We write $\mathsf{Sep}(Q, D, \bar{a}, \bar{b})$ (separating $\bar{a}$ and $\bar{b}$ with respect to Q and D) for the statement $\mathsf{Supp}(Q, D, \bar{a}) - \mathsf{Supp}(Q, D, \bar{b}) \neq \emptyset$. Then $\bar{a} \trianglelefteq_{Q,D} \bar{b}$ iff $\mathsf{Sep}(Q, D, \bar{a}, \bar{b})$ is false, and $\bar{a} \triangleleft_{Q,D} \bar{b}$ iff $\mathsf{Sep}(Q, D, \bar{a}, \bar{b})$ is false, and $\mathsf{Sep}(Q, D, \bar{b}, \bar{a})$ is true. Note that $\mathsf{Sep}(Q, D, \bar{a}, \bar{b})$ is trivially in NP: it suffices to guess a valuation v and then, using polynomial time data complexity of Q, check whether $v(\bar{a}) \in Q(v(D))$ and $v(\bar{b}) \notin Q(v(D))$. This gives us membership of $\trianglelefteq$-Comparison and $\triangleleft$-Comparison in coNP and DP respectively.

To prove coNP-hardness of $\trianglelefteq$-Comparison, we reduce from graph non-3-colorability. For DP-hardness, we reduce from the problem of checking whether $\chi(G)$, the chromatic number of G, equals 4, which is known to be DP-complete [38]. □

Next we analyze the complexity of BestAnswer. Compared to $\trianglelefteq$ and $\triangleleft$, it goes up a bit but still stays in the second level of the polynomial hierarchy.

Theorem 7. *For first-order queries, the problem* BestAnswer *is $\mathrm{P}^{\mathrm{NP}[\log n]}$-complete in data complexity. It remains in $\mathrm{P}^{\mathrm{NP}[\log n]}$ for queries with polynomial time data complexity.*

Proof sketch. To prove membership, we use the description of $\mathrm{P}^{\mathrm{NP}[\log n]}$ as the class of problems solvable in polynomial time with parallel access to an NP-oracle, see [11]. Suppose we are given a k-ary query Q of polynomial data complexity and a database D. Since Q is fixed, in polynomial time we can enumerate all the pairs $(\bar{a}, \bar{b})$ of k-tuples of $\mathrm{adom}(D)$. Then, in parallel, we can check for each pair whether $\mathsf{Sep}(Q, D, \bar{a}, \bar{b})$ holds, using an NP oracle. With this information, in quadratic time we know whether $\bar{a} \trianglelefteq_{Q,D} \bar{b}$ and whether $\bar{a} \triangleleft_{Q,D} \bar{b}$ for all such pairs. With this, again in quadratic time, we can find the set $\mathsf{Best}(Q, D)$.

To prove hardness, we reduce from the following problem: given an undirected graph G, is its chromatic number $\chi(G)$ odd? This problem is known to be $\mathrm{P}^{\mathrm{NP}[\log n]}$-complete [44]. □

5.1 Unions of conjunctive queries

How can we lower the complexity of tuple comparisons with respect to $\trianglelefteq$ and $\triangleleft$? Checking whether $\bar{a} \trianglelefteq_{Q,D} \bar{b}$ is already coNP-hard for unions of conjunctive queries with negation (this follows from the proof of Theorem 6); thus it seems that unions of conjunctive queries, without negation, is a natural candidate. At first this does not look surprising, since computing certain answers to unions of conjunctive queries can be done by naïve evaluation and

thus has polynomial time data complexity. However, naïve evaluation is of no help with the problems we are considering here. Consider, for instance, the $\trianglelefteq$ ordering. It is easy to see that $\bar{a} \trianglelefteq_{Q,D} \bar{b}$ iff $Q(\bar{a}) \rightarrow Q(\bar{b})$ is certainly true (viewing $Q(\bar{a}) \rightarrow Q(\bar{b})$ as a Boolean query saying that $Q(\bar{a})$ implies $Q(\bar{b})$). But naïve evaluation of this query does not tell us whether $\bar{a} \trianglelefteq_{Q,D} \bar{b}$. Indeed, let D be a database with relation $R = \{(1, \bot), (\bot, 2)\}$, and let Q return R. Then for $\bar{a} = (1, 2)$ and $\bar{b} = (1, 1)$, the naïve evaluation of $Q(\bar{a}) \rightarrow Q(\bar{b})$ returns *true* as neither tuple is in R, but of course $\bar{a} \trianglelefteq_{Q,D} \bar{b}$ is false, since $\mathrm{Supp}(Q, D, \bar{a}) = \{\bot \mapsto 1, \bot \mapsto 2\}$ and $\mathrm{Supp}(Q, D, \bar{b}) = \{\bot \mapsto 1\}$.

Thus, one needs an approach different from the usual naïve evaluation. The previous observation offers a suggestion: checking certainty of implication of conjunctive queries is known to be solvable in polynomial time [21]. We can extend those techniques to show the following result.

Theorem 8. *For unions of conjunctive queries, all of* $\trianglelefteq$-Comparison, $\triangleleft$-Comparison, *and* BestAnswer *have polynomial time data complexity.*

Proof sketch. Recall that $\mathrm{Sep}(Q, D, \bar{a}, \bar{b})$ stands for the statement $\mathrm{Supp}(Q, D, \bar{a}) - \mathrm{Supp}(Q, D, \bar{b}) \neq \emptyset$. Since knowing truth values of $\mathrm{Sep}(Q, D, \bar{a}, \bar{b})$ and $\mathrm{Sep}(Q, D, \bar{b}, \bar{a})$ suffices to determine whether $\bar{a} \trianglelefteq_{Q,D} \bar{b}$ and $\bar{a} \triangleleft_{Q,D} \bar{b}$, and for a fixed k-ary query Q there are polynomially many k-tuples of the active domain, it suffices to check that $\mathrm{Sep}(Q, D, \bar{a}, \bar{b})$ can be solved in polynomial time if Q is fixed.

Let D have m nulls, and let Q be C-generic. Let A_m be a set of m constants that is disjoint from $\mathrm{Const}(D)$ and C. Then, if $\mathrm{Sep}(Q, D, \bar{a}, \bar{b})$ is true, we can find a valuation $v \in \mathrm{Supp}(Q, D, \bar{a}) - \mathrm{Supp}(Q, D, \bar{b})$ such that $\mathrm{range}(v) \subseteq \mathrm{Const}(D) \cup C \cup A_m$. Indeed, assume we found some valuation $v' \in \mathrm{Supp}(Q, D, \bar{a}) - \mathrm{Supp}(Q, D, \bar{b})$, and its range contains $l \leq m$ elements $c'_1, \ldots, c'_l$ that are not in $\mathrm{Const}(D) \cup C \cup A_m$. Since $|A_m| = m$, we can find l elements $c_1, \ldots, c_l$ of A_m that are not in $\mathrm{range}(v')$, and define v as $\pi \circ v'$, where π sends each c'_i to c_i for $i \leq l$. Then $\mathrm{range}(v) \subseteq \mathrm{Const}(D) \cup C \cup A_m$. Moreover, by C-genericity, $v(\bar{a}) \in Q(v(D))$ iff $v'(\bar{a}) \in Q(v'(D))$ and likewise for $\bar{b}$, and hence $v \in \mathrm{Supp}(Q, D, \bar{a}) - \mathrm{Supp}(Q, D, \bar{b})$. Thus, we set $A = \mathrm{Const}(D) \cup C \cup A_m$, which is linear in the size of D (since C is fixed). It then suffices to show that in polynomial time, in the size of D, we can check whether there exists a valuation $v \in \mathrm{Supp}(Q, D, \bar{a}) - \mathrm{Supp}(Q, D, \bar{b})$ such that $\mathrm{range}(v) \subseteq A$. We call this a (D, A)-separating valuation (since $\bar{a}$ and $\bar{b}$ are fixed, we do not mention them explicitly).

Let $Q(\bar{x})$ be the union of k-ary conjunctive queries $Q_1, \ldots, Q_r$, where each $Q_i(\bar{x})$ is of the form $\exists \bar{y}\, \alpha_1(\bar{x}, \bar{y}) \wedge \ldots \wedge \alpha_{p_i}(\bar{x}, \bar{y})$, and all $\alpha_j(\bar{x}, \bar{y})$s are relational atoms. Let $p = \max_{i \leq r} p_i$. Then we prove that the following are equivalent:

- (*) there exists a (D, A)-separating valuation;
- (**) there is a subset $D' \subseteq D$ with at most $p + k$ tuples whose active domain contains all the components of $\bar{a}$, and a valuation v' on D' with $\mathrm{range}(v') \subseteq A$ that satisfies $v'(\bar{a}) \in Q(v'(D'))$ and $v'(\bar{b}) \notin Q^{\text{naïve}}(v'(D))$.

Note that v' is only defined on the nulls of D', not of D, and thus $v'(D)$ and $v'(\bar{b})$ may still contain nulls, so it is appropriate to talk about naïve evaluation of Q on $v'(D)$.

Since we know that checking $\mathrm{Sep}(Q, D, \bar{a}, \bar{b})$ amounts to checking (*), the equivalence tells us that we can check (**) instead. This is easily seen to be in polynomial time with respect to data complexity. □

5.2 Combining qualitative and quantitative

Results established so far classify query answers into good and bad along two different axes. Qualitatively, we want to find answers that are in $\mathrm{Best}(Q, D)$ and discard others. According to quantitative measures, good answers are those that are almost certainly true (i.e., $\mu(Q, D, \bar{a}) = 1$), and bad ones are those that are almost certainly false (i.e., $\mu(Q, D, \bar{a}) = 0$). Is there any relationship between these two classifications? We show below that they are completely orthogonal: all four options among best vs non-best and almost certainly true vs false are possible.

Proposition 7. *All four possibilities of best vs non-best and almost certainly true vs almost certainly false are realizable for first-order queries.*

Proof sketch. Consider a database D with unary relations A and B holding distinct elements a and b respectively, and a binary relation R with a single tuple $(\bot, \bot')$. Define

$$Q(x) \;=\; \Big(B(x) \wedge \exists x\, R(x, x)\Big) \;\vee\; \Big(A(x) \wedge \neg \exists x\, R(x, x)\Big)$$

Then $\mathrm{Supp}(Q, D, a)$ has valuations v with $v(\bot) \neq v(\bot')$ and $\mathrm{Supp}(Q, D, b)$ has valuations v with $v(\bot) = v(\bot')$. In particular $\mathrm{Best}(Q, D) = \{a, b\}$. Furthermore, $\mu^k(Q, D, a) = |\mathrm{Supp}^k(Q, D, a)|/|\mathrm{V}^k(D)| = 1 - 1/k$ and likewise $\mu^k(Q, D, b) = |\mathrm{Supp}^k(Q, D, b)|/|\mathrm{V}^k(D)| = 1/k$, and thus $\mu(Q, D, a) = 1$ and $\mu(Q, D, b) = 0$. This shows that options (best, μ=0) and (best, $\mu = 1$) are realizable.

Expand D with a new relation G containing a single element g, and let $Q'(x)$ be $G(x) \vee Q(x)$. For this new database D' we have $\mathrm{Best}(Q', D') = \{g\}$ and thus $a, b \notin \mathrm{Best}(Q', D')$, but we still have $\mu(Q, D', b) = 1$ and $\mu(Q, D', a) = 0$, showing that the options of non-best and $\mu = 1$ or $\mu = 0$ are realizable as well. □

Also, looking at qualitative and quantitative notions together suggests that it would make sense to restrict $\mathrm{Best}(Q, D)$ to almost certainly true answers, to get the best of both worlds. That is, we define $\mathrm{Best}_\mu(Q, D)$ as the set $\{\bar{a} \mid \bar{a} \in \mathrm{Best}(Q, D) \text{ and } \mu(Q, D, \bar{a}) = 1\}$. This gives rise to a modified problem BestAnswer_μ which is defined just as BestAnswer except that we want to check whether $\mathrm{Best}_\mu(Q, D)$ is in $\mathcal{X}$. We can show that the complexity of this modified version is exactly the same as for BestAnswer itself.

Proposition 8. *The problem* BestAnswer_μ *remains* $\mathrm{P}^{\mathrm{NP}[\log n]}$*-complete for first-order queries, and has polynomial-time data complexity for unions of conjunctive queries.* □

6 FURTHER DIRECTIONS

Having introduced a new framework for measuring and comparing certainty of query answers over incomplete databases, we now outline its possible future applications and extensions.

SQL nulls. We used the common model of marked nulls. Its variant of non-repeating nulls, known as Codd nulls, is often used as a simplified model of SQL nulls. Nonetheless, SQL nulls are not

Codd nulls. Crucially, they follow a 3-valued logic in query evaluation [15]; there are other more subtle differences that arise when one compares nulls. While the results shown here are applicable to marked and Codd nulls, we would like to know how to interpret them with SQL nulls used in all relational DBMSs.

Quality of Approximations. It has long been known that finding certain answers is computationally hard even for relational algebra queries [2]. Due to this, various techniques for approximating certain answers were proposed. First approaches appeared long ago [37, 43], but more recently they were reworked for all relational algebras queries and shown to behave well when implemented in commercial RDBMSs [26, 32]. One question that remained open though is the quality of such approximations. While it was experimentally measured in [26], the only theoretical guarantee we have is that on databases without nulls, approximation schemes do not lose any answers. We would like to use the techniques developed here to measure the quality of queries approximating certain answers, by measuring the likelihood of a certain answer not being returned by the approximating query.

Preferences. In defining the measure $\mu(Q, D)$ in the absence of constraints we assumed that every constant value is equally likely to be substituted for a null. But there are many scenarios where this is not so, and some values are preferred to others. For example, if $\perp$ stands for the disease of a particular patient in a database, we may have additional information on the likelihood of different diagnoses. If this information is given probabilistically, we have essentially the model of probabilistic databases [40]. But it is equally, if not more likely, that such information will be given in terms of preferences of some values over others. Defining preferences, computing and reasoning with them is of course an important subject in AI [16], and we would like to bring in preferences to reason about query answers over incomplete data using the framework defined here.

Other distributions. Related to the previous point, the model without constraints was essentially that of uniform distributions: every value can be assigned to a null equally likely. In the theory of random graphs and structures, one studies distributions more complex than uniform, often depending on the number of nodes of a graph; a typical example is flipping a coin with probability $n^{-\alpha}$ to put an edge between two of n nodes, where $\alpha \in (0, 1)$, see, e.g., [39]. Looking at other distributions and the behavior of the measures $\mu(Q, D)$ and $\mu(Q|\Sigma, D)$ under them is another possible direction.

Numerical and ordered domains. We assumed that domain elements are uninterpreted, in which case the notion of genericity makes perfect sense for queries. But if databases store elements that can be ordered, or numbers, and ordering/arithmetic operations appear in queries, more refined notions of genericity, and in fact of the measures of certainty, are needed. Finding such notions may shed light on answering queries on incomplete databases with interpreted elements. This is the area we know little about – but we do know that even small additions such as order lead to a significant increase in complexity [41].

Other semantics and applications. For this initial study we concentrated on the very common closed-world semantics of missing values, but other interpretations of incompleteness and other semantics exist [1, 27, 42]. We touched briefly on the open-world semantics, but the negative result we showed does not mean that there are no alternative ways of measuring certainty in a meaningful way. In addition to applying the measure-based approach to other semantics, we would like to investigate its suitability for query answering in applications that rely on certain answers, such as data integration, data exchange, OBDA, and consistent query answering, where probabilistic approaches have already been considered [24].

Acknowledgments. I am grateful to Paolo Guagliardo and Evgenia Ternovska for discussions that led to the questions answered in this paper, and to all who provided comments on earlier drafts: Pablo Barceló, Marco Calautti, Marco Console, Ron Fagin, Andreas Pieris, Juan Reutter, and the reviewers. Work partially supported by EPSRC grants M025268 and N023056.

REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[2] 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.
[3] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.
[4] M. Arenas, J. Pérez, and J. L. Reutter. Data exchange beyond complete data. *Journal of the ACM*, 60(4):28:1–28:59, 2013.
[5] M. Arenas and M. Ugarte. Designing a query language for RDF: marrying open and closed worlds. In *PODS*, pages 225–236, 2016.
[6] S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.
[7] P. Barceló, L. Libkin, and M. Romero. Efficient approximations of conjunctive queries. *SIAM J. Comput.*, 43(3):1085–1130, 2014.
[8] L. Bertossi. *Database Repairing and Consistent Query Answering*. Morgan&Claypool Publishers, 2011.
[9] M. Bienvenu and M. Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Reasoning Web*, pages 218–307, 2015.
[10] A. Blass, Y. Gurevich, and D. Kozen. A zero-one law for logic with a fixed-point operator. *Information and Control*, 67(1-3):70–90, 1985.
[11] S. R. Buss and L. Hay. On truth-table reducibility to SAT. *Information and Computation*, 91(1):86–102, 1991.
[12] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Data integration under integrity constraints. *Informatiom Systems*, 29(2):147–163, 2004.
[13] 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.
[14] K. Compton. Some useful preservation theorems. *Journal of Symbolic Logic*, 48(2):427–440, 1983.
[15] C. J. Date and H. Darwen. *A Guide to the SQL Standard*. Addison-Wesley, 1996.
[16] C. Domshlak, E. Hüllermeier, S. Kaci, and H. Prade. Preferences in AI: an overview. *Artificial Intelligence*, 175(7-8):1037–1052, 2011.
[17] R. Fagin. Probabilities on finite models. *Journal of Symbolic Logic*, 41(1):50–58, 1976.
[18] R. Fagin, P. Kolaitis, and L. Popa. Data exchange: getting to the core. *ACM TODS*, 30(1):174–210, 2005.
[19] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theoretical Computer Science*, 336:89–124, 2005.
[20] B. Fazzinga, S. Flesca, and F. Parisi. On the complexity of probabilistic abstract argumentation frameworks. *ACM TOCL*, 16(3):22:1–22:39, 2015.
[21] A. Gheerbrant and L. Libkin. Certain answers over incomplete XML documents: Extending tractability boundary. *Theory Comput. Syst.*, 57(4):892–926, 2015.
[22] A. Gheerbrant, L. Libkin, and C. Sirangelo. Naïve evaluation of queries over incomplete databases. *ACM TODS*, 39(4):31:1–31:42, 2014.
[23] E. Grandjean. Complexity of the first-order theory of almost all finite structures. *Information and Control*, 57(2/3):180–204, 1983.
[24] S. Greco and C. Molinaro. Approximate probabilistic query answering over inconsistent databases. In *ER*, pages 311–325, 2008.
[25] S. Greco, C. Molinaro, and F. Spezzano. *Incomplete Data and Data Dependencies in Relational Databases*. Morgan & Claypool Publishers, 2012.

[26] P. Guagliardo and L. Libkin. Making SQL queries correct on incomplete databases: A feasibility study. In *PODS*, pages 211–223, 2016.
[27] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.
[28] P. G. Kolaitis and M. Y. Vardi. Infinitary logics and 0-1 laws. *Information and Computation*, 98(2):258–294, 1992.
[29] P. G. Kolaitis and M. Y. Vardi. 0-1 laws for fragments of existential second-order logic: A survey. In *MFCS*, pages 84–98, 2000.
[30] M. Lenzerini. Data integration: a theoretical perspective. In *PODS*, pages 233–246, 2002.
[31] L. Libkin. *Elements of Finite Model Theory*. Springer-Verlag, 2004.
[32] L. Libkin. SQL's three-valued logic and certain answers. *ACM TODS*, 41(1):1:1–1:28, 2016.
[33] W. Lipski. On relational algebra with marked nulls. In *PODS*, pages 201–203, 1984.
[34] J. Lynch. Almost sure theories. *Ann. Math. Logic*, 18:91–135, 1980.
[35] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.
[36] R. Reiter. On closed world data bases. In *Logic and Data Bases*, pages 55–76, 1977.
[37] R. Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *Journal of the ACM*, 33(2):349–370, 1986.
[38] J. Rothe. Exact complexity of exact-four-colorability. *Information Processing Letters*, 87(1):7–12, 2003.
[39] J. Spencer. *The Strange Logic of Random Graphs*. Springer, 2001.
[40] D. Suciu, D. Olteanu, C. Re, and C. Koch. *Probabilistic Databases*. Morgan&Claypool Publishers, 2011.
[41] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *JCSS* 54(1): 113–135 (1997).
[42] R. van der Meyden. Logical approaches to incomplete information: A survey. In *Logics for Databases and Information Systems*, pages 307–356, 1998.
[43] M. Vardi. Querying logical databases. *JCSS*, 33(2):142–160, 1986.
[44] K. W. Wagner. Bounded query classes. *SIAM J. Comput.*, 19(5):833–846, 1990.