Foundations of Query Answering on Inconsistent Databases

Jef Wijsen
University of Mons
Mons, Belgium
jef.wijsen@umons.ac.be

ABSTRACT

Notwithstanding the traditional view that database instances must respect all integrity constraints imposed on them, it is relevant to develop theories about how to handle database instances that violate some integrity constraints, and more particularly, how to cope with query answering in the presence of inconsistency. Such a theory developed over the past twenty years is currently known as *consistent query answering* (CQA). The aim of this article is to summarize and discuss some core concepts and theoretical developments in CQA.

1. INTRODUCTION

Consistent query answering (CQA) started with an article at PODS 1999 by Arenas, Bertossi and Chomicki [2]. Twenty years later, the significance of their contribution was acknowledged through a *Gems of PODS* session, at which occasion Leopoldo Bertossi flashed back to the origins of CQA [6]. Among these origins is the question about "what is consistent in an inconsistent database" [6, p. 49]. The next example illustrates this question, as well as the answer provided by the CQA approach.

Following the well-known running example of C.J. Date's textbook [15], we represent suppliers by a table with name S. Every supplier has a supplier number, unique to that supplier, a supplier name, a rating or status value, and a location. The requirement that supplier numbers be unique is violated by the following table; the available information about the status of S2 happens to be inconsistent.

S	S#	SNAME	STATUS	CITY
	S1	Smith	20	London
	S2	Jones	10	Paris
	S2	Jones	15	Paris

We ask the question of what information can and cannot be inferred from this inconsistent table. A useful answer to that question should rely on some *paraconsistent* inference relation, i.e., one that abandons the principle that "everything follows from an inconsistency" (*ex contradictione quodlibet*). In general terms, the answer provided in [2] goes as follows. Every inconsistent database instance represents a set of possible consistent database instances, which are called *repairs*. Repairs should be obtained by fixing inconsistencies in some minimal way. *Consistently true* information, then, is defined as information that holds true in every repair.

We will formalize shortly the idea of minimal fixing. For the example table, minimal fixing could mean deleting either of the two tuples that agree on S#. This yields two repairs which differ in the status of supplier S2. Other repairs could be obtained by replacing either occurrence of S2 with some fresh supplier number. This would yield many repairs, differing only in the choice of the new supplier number. For the purpose of this example, assume that there are no other repairs than the ones just described. Then, the assertion that "S2 is a supplier having a lower status than S1" is consistently true, because it is true in every repair. It is also consistently true that "the number of suppliers is two or three." On the other hand, the claim that "supplier S2 has status 10" is not consistently true.

The aim of this article is to present and discuss theoretical foundations of CQA, with an emphasis on results in computational and descriptive complexity. We will not report on the deployment of CQA in operational systems. We have tried to be complimenatry to the previously cited *Gems of PODS* article of Bertossi [6], which gives a broad overview, intentionally based on representative examples rather than formal definitions. The treatment in the current paper is more formal and focuses on some core concepts and results.

Organization.

Section 2 recalls some standard notions in database theory. Section 3 formalizes the framework of consistent query answering. The idea of minimal fixing is introduced there in an original manner especially developed for the purpose of this article. Section 4 discusses different ways for indicating the complexity of consistent query answering, referring to both computational and descriptive complexity. In Section 5, we focus on a fine-grained complexity classification that has been achieved for consistent query answering with respect to primary keys, but which is largely open for other classes of integrity constraints. The case of primary keys is also interesting because of its connections to two other fields: constraint satisfaction problems (CSPs) and probabilistic database systems. Section 6 discusses the complexity of repair checking. Finally, Section 7 concludes the paper.

2. PRELIMINARIES

For every positive integer n, we assume denumerably many *relation names* of *arity* n. A *database schema* is a finite set of relation names. In the sequel, we will often assume that some database schema has been fixed. A *database instance* is a finite set of *facts* $R(c_1, \ldots, c_n)$ where R is an n-ary relation name of the database schema, and each c_i is a constant. An m-ary *query* q, with $m \geq 0$, maps every database instance $\mathbf{db}$ to an m-ary relation, denoted $q(\mathbf{db})$. A 0-ary query is also called a Boolean query: a 0-ary relation represents *false* if it is empty, otherwise it represents *true*. In the complexity study of CQA, much attention has been paid to Boolean conjunctive queries, i.e., queries defined by first-order logic sentences, possibly with constants, of the form

$$\exists \vec{x} \, (R_1(\vec{x}_1) \wedge R_2(\vec{x}_2) \wedge \cdots \wedge R_n(\vec{x}_n)) \, .$$

Such a query is said to be *self-join-free* if $i \neq j$ implies $R_i \neq R_j$. Each $R_i(\vec{x}_i)$ is called a *relational atom*.

We assume that our database schema is equipped with a set Σ of *integrity constraints*. All integrity constraints in this paper can be expressed as sentences in first-order logic. In what follows, by a *set of integrity constraints*, we will always mean a finite set of integrity constraints that can be satisfied by some database instance. A database instance is *consistent* if it satisfies all integrity constraints in Σ; otherwise it is *inconsistent*. Common integrity constraints, called *dependencies*, are recalled next. Readers familiar with common classes of integrity constraints can skip the remainder of this section.

Inclusion dependencies (IND).

If R and S are relation names, of arities m and n respectively, then $R[i_1, \ldots, i_k] \subseteq S[j_1, \ldots, j_k]$ is an *inclusion dependency*, where $i_1, \ldots, i_k$ is a sequence of distinct integers in $\{1, \ldots, m\}$, and $j_1, \ldots, j_k$ is a sequence of distinct integers in $\{1, \ldots, n\}$. A database instance $\mathbf{db}$ is said to *satisfy* this inclusion dependency if for every $R(a_1, \ldots, a_m) \in \mathbf{db}$, there exists $S(b_1, \ldots, b_n) \in \mathbf{db}$ such that for every $\ell \in \{1, \ldots, k\}$, $a_{i_\ell} = b_{j_\ell}$.

Functional dependencies (FD).

If R is a relation name of arity n, then a *functional dependency* is an expression $R : X \rightarrow Y$ where $X, Y \subseteq \{1, \ldots, n\}$. A database instance $\mathbf{db}$ *satisfies* $R : X \rightarrow Y$ if for all $R(a_1, \ldots, a_n)$, $R(b_1, \ldots, b_n) \in \mathbf{db}$, if $a_i = b_i$ for all $i \in X$, then $a_j = b_j$ for all $j \in Y$. If $X \cup Y = \{1, \ldots, n\}$, then $R : X \rightarrow Y$ is called a *key dependency.*

Tuple-generating dependencies (tgd).

A $\vee$-*tgd* is a constant-free first-order logic sentence of the form

$$\forall \vec{x} \left(\varphi(\vec{x}) \rightarrow \bigvee_{i=1}^{n} \exists \vec{y_i} \psi_i(\vec{x}, \vec{y_i}) \right), \qquad (1)$$

where φ is a nonempty conjunction of relational atoms, each ψ_i is a conjunction of relational atoms, and every variable in $\vec{x}$ appears in φ (but not necessarily in $\bigvee_{i=1}^{n} \exists \vec{y_i} \psi_i(\vec{x}, \vec{y_i})$). $\vee$-tgds can be further restricted as follows:

- a *tgd* is a $\vee$-tgd where $n = 1$;
- a $\vee$-tgd or tgd without existentially-quantified variables is called *full*; and
- a *LAV tgd* is a tgd of the form

  $$\forall \vec{x} \, (R(\vec{x}) \rightarrow \exists \vec{y} \psi(\vec{x}, \vec{y})) \, ,$$

  where $R(\vec{x})$ is a relational atom.

For a set of tgds, the notion of being *weakly acyclic* is a structural property that guarantees termination of the *chase*. The definition of weakly acyclic can be found in [17].

Universal constraints (UC).

A *universal constraint* is a constant-free first-order logic sentence of the form

$$\forall \vec{x} \left(\varphi(\vec{x}) \wedge \beta(\vec{x}) \rightarrow \bigvee_{i=1}^{n} \psi_i(\vec{x}) \right), \qquad (2)$$

where φ and each ψ_i are conjunctions of relational atoms, β is a Boolean combination of equalities, and every variable in $\vec{x}$ appears in φ.

Special cases of UCs are obtained by letting $n = 0$, and by considering that the empty disjunction is **false**:

- a *denial constraint* is commonly written in the form $\forall\vec{x}\neg\left(\varphi(\vec{x})\wedge\beta(\vec{x})\right)$, a sentence logically equivalent to $\forall\vec{x}\left(\varphi(\vec{x})\wedge\beta(\vec{x})\rightarrow\mathbf{false}\right)$.

- an *equality-generating dependency (egd)* takes the form $\forall\vec{x}\left(\varphi(\vec{x})\rightarrow x_i = x_j\right)$, which is equivalent to $\forall\vec{x}\left(\varphi(\vec{x})\wedge\neg\left(x_i = x_j\right)\rightarrow\mathbf{false}\right)$.

The form (2) was chosen because of its resemblance to (1). The disjunction $\bigvee_{i=1}^{n}\psi_i(\vec{x})$ in (2) is equivalent to a formula in CNF, say $\bigwedge_{i=1}^{m}\chi_i(\vec{x})$, where each χ_i is a disjunction of relational atoms. The set $\{\forall\vec{x}\left(\varphi(\vec{x})\wedge\beta(\vec{x})\rightarrow\chi_i(\vec{x})\right)\}_{i=1}^{m}$ is then equivalent to (2). Since we always consider sets of integrity constraints, universal constraints can be (and often are) defined in different forms [2, 31]:

$$\forall\vec{x}\neg\left(\begin{array}{l}\neg R_1(\vec{x}_1)\wedge\cdots\wedge\neg R_m(\vec{x}_m)\wedge\\ R_{m+1}(\vec{x}_{m+1})\wedge\cdots\wedge R_n(\vec{x}_n)\wedge\beta(\vec{x})\end{array}\right),$$

$$\forall\vec{x}\left(\begin{array}{l}R_1(\vec{x}_1)\vee\cdots\vee R_m(\vec{x}_m)\vee\\ \neg R_{m+1}(\vec{x}_{m+1})\vee\cdots\vee\neg R_n(\vec{x}_n)\vee\neg\beta(\vec{x})\end{array}\right),$$

$$\forall\vec{x}\left(\begin{array}{l}R_{m+1}(\vec{x}_{m+1})\wedge\cdots\wedge R_n(\vec{x}_n)\wedge\beta(\vec{x})\rightarrow\\ R_1(\vec{x}_1)\vee\cdots\vee R_m(\vec{x}_m)\end{array}\right),$$

where each R_i is a relation name, and each variable in $\vec{x}$ appears in some $\vec{x}_i$ with $m+1\leq i\leq n$. The latter requirement is called *safety*. Denial constraints, then, are universal constraints where $m = 0$.

Figure 1 relates different classes of integrity constraints. An upward line from IC_1 to IC_2 means that every set Σ_1 of integrity constraints in the class IC_1 is equivalent to some set Σ_2 in IC_2. Note, for example, that the functional dependency $R : \{1\}\rightarrow\{2,3\}$ expresses a pair of egds.

3. CONSISTENT ANSWERS

In Section 1, we have already given a general but informal introduction to CQA. We will now enter into more technical details.

3.1 Querying Inconsistent Data

Let q be a query and Σ a set of integrity constraints. If $\mathbf{db}$ is a consistent database instance, the *query answer* $q(\mathbf{db})$ can reasonably be called "consistent" as well. The CQA paradigm was developed in the first place to define "consistent query answers" for the case where $\mathbf{db}$ is inconsistent.

When $\mathbf{db}$ is an inconsistent database instance, it looks like a good idea to change it, in some minimal way, such that the new database instance is consistent. Such a consistent database instance obtained by some minimal change is called a *repair*. Let us skip for a moment the details of minimal change, and denote by $\mathsf{repairs}(\mathbf{db},\Sigma)$ the set of all repairs of $\mathbf{db}$ with respect to Σ. Then a tuple t belongs to the consistent answer to q on $\mathbf{db}$ if for each repair $\mathbf{r}$, we have that t belongs to $q(\mathbf{r})$:

$$\mathsf{cqa}(q,\mathbf{db},\Sigma)\triangleq\bigcap\{q(\mathbf{r})\mid\mathbf{r}\in\mathsf{repairs}(\mathbf{db},\Sigma)\}.$$

Of course, this definition makes sense only when the notion of being a repair is well-defined, which is the topic of the next subsection.

3.2 Fixing Inconsistency

The literature on CQA contains many proposals for formalizing the idea of minimal change. For the purpose of this article, we next develop a generalization that captures the essence of most proposals. Our generalization assumes that, for a given database instance $\mathbf{db}$, there is a binary relation $\leq_{\mathbf{db}}$ (which depends on $\mathbf{db}$) on the set of all consistent database instances. The intended informal meaning is that for all consistent database instances $\mathbf{r}_1$ and $\mathbf{r}_2$, we have $\mathbf{r}_1\leq_{\mathbf{db}}\mathbf{r}_2$ if transforming $\mathbf{db}$ into $\mathbf{r}_1$ requires not more effort than transforming $\mathbf{db}$ into $\mathbf{r}_2$. We define the strict version of $\leq_{\mathbf{db}}$, denoted $<_{\mathbf{db}}$, as follows: $\mathbf{r}_1<_{\mathbf{db}}\mathbf{r}_2\triangleq\mathbf{r}_1\leq_{\mathbf{db}}\mathbf{r}_2$ and not $\mathbf{r}_2\leq_{\mathbf{db}}\mathbf{r}_1$. The principle that repairs must be obtained by some minimal change can now be made formal: a *repair* of $\mathbf{db}$ is a consistent database instance $\mathbf{r}$ such that there exists no consistent database instance $\mathbf{r}'$ satisfying $\mathbf{r}'<_{\mathbf{db}}\mathbf{r}$. Some repairs are of a special sort: a repair of $\mathbf{db}$ is called a *subset-repair* if it is included in $\mathbf{db}$, and is called a *superset-repair* if it includes $\mathbf{db}$.

To guarantee the existence of repairs, some additional properties should be imposed on $\leq_{\mathbf{db}}$ (or on $<_{\mathbf{db}}$). A very common and sufficient requirement for the existence of repairs is the acyclicity of $<_{\mathbf{db}}$.

In the CQA literature, the binary relation $\leq_{\mathbf{db}}$ (or $<_{\mathbf{db}}$) is never explicitly given, but instead implicitly specified. Different specifications of $\leq_{\mathbf{db}}$ now lead to different repair notions; the two most common are the following:

- Let $\oplus$ denote the symmetric difference between sets. When we define $\mathbf{r}_1\leq_{\mathbf{db}}\mathbf{r}_2\triangleq(\mathbf{r}_1\oplus\mathbf{db})\subseteq(\mathbf{r}_2\oplus\mathbf{db})$, we obtain *symmetric-difference repairs*, also called *$\oplus$-repairs*. With this definition, $\leq_{\mathbf{db}}$ is a partial order. In terms of minimal change, symmetric-difference repairs minimize (with respect to $\subseteq$) the set of inserted and deleted facts.

- When we define $\mathbf{r}_1\leq_{\mathbf{db}}\mathbf{r}_2\triangleq|\mathbf{r}_1\oplus\mathbf{db}|\leq|\mathbf{r}_2\oplus\mathbf{db}|$, we obtain *cardinality repairs*, also called *C-repairs*. With this definition, $\leq_{\mathbf{db}}$ is reflexive and transitive.

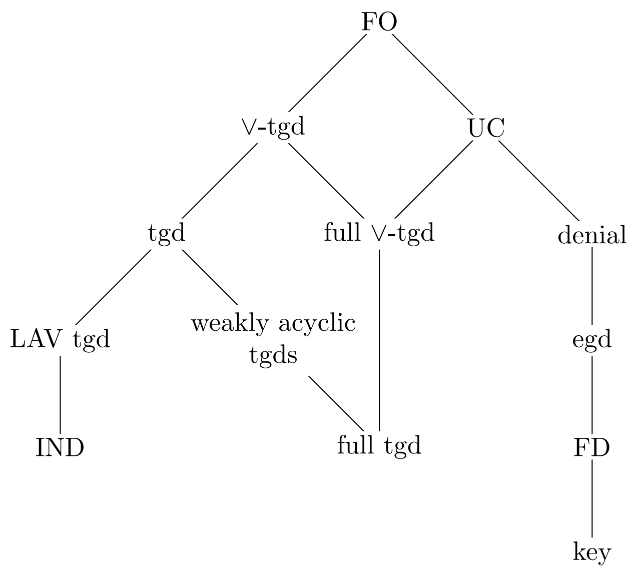

Figure 1: Common classes of integrity constraints. Adapted from [4].

In the literature on CQA, the notion of subset-repair is frequently introduced as an independent repair notion—one in which deletions are the only allowed repairing actions. We, instead, have defined it as a property: a $\oplus$-repair, cardinality repair, or any other repair of a database instance $\mathbf{db}$ is called a subset-repair if it is included in $\mathbf{db}$. Clearly, every maximal (with respect to $\subseteq$) consistent subset of a database instance $\mathbf{db}$ is a $\oplus$-repair of $\mathbf{db}$. A significant observation (see [1, Proposition 3]) is that for sets of denial constraints, every $\oplus$-repair is a subset-repair. This observation is no longer true for universal constraints, as illustrated next.

Example 1. *The database instance $\{R(a)\}$ has two $\oplus$-repairs with respect to $\Sigma = \{R(a) \rightarrow S(b)\}$: $\{\}$ and $\{R(a), S(a)\}$, which are a subset-repair and a superset-repair, respectively.*

In the definition of $\oplus$-repairs, the symmetric difference $\mathbf{r}_1 \oplus \mathbf{db}$ treats deletions (i.e., facts in $\mathbf{db} \setminus \mathbf{r}_1$) and insertions (i.e., facts in $\mathbf{r}_1 \setminus \mathbf{db}$) on equal footing. Alternatively, it is conceivable, although less common, to treat deletions and insertions asymmetrically. In some situations, for example, inconsistency may be primarily attributed to missing facts rather than to erroneously stored facts. In such situations, one may want to keep deletions to a minimum, which can be achieved by defining $\mathbf{r}_1 \leq_{\mathbf{db}} \mathbf{r}_2 \triangleq$ either $(\mathbf{r}_1 \cap \mathbf{db}) \supsetneq (\mathbf{r}_2 \cap \mathbf{db})$ or both $(\mathbf{r}_1 \cap \mathbf{db}) = (\mathbf{r}_2 \cap \mathbf{db})$ and $(\mathbf{r}_1 \oplus \mathbf{db}) \subseteq (\mathbf{r}_2 \oplus \mathbf{db})$. With this definition, the repair $\{R(a), S(a)\}$ would become the only repair in Example 1, because it preserves the database fact $R(a)$. Many other repair notions have been proposed and studied over the past twenty years.

Prioritized repairs.

The framework introduced in [32] and further investigated in [16, 20, 24] deduces $<_{\mathbf{db}}$ from a preference relation on the set of database facts. For the sake of simplicity, let Σ be a set of functional dependencies. An *inconsistent prioritizing database instance* is a pair $(\mathbf{db}, \succ)$ where $\mathbf{db}$ is an inconsistent database instance, and $\succ$ is an acyclic binary relation on $\mathbf{db}$ such that if $f \succ g$, then $\{f, g\}$ falsifies Σ. The intended informal meaning of $f \succ g$ is that we prefer to keep f rather than g. Note that in the case of functional dependencies, conflicting facts always come in pairs. A relation $<_{\mathbf{db}}$ can be defined in terms of $\succ$, as follows: for all distinct consistent subsets $\mathbf{r}_1$ and $\mathbf{r}_2$ of $\mathbf{db}$, define $\mathbf{r}_1 <_{\mathbf{db}} \mathbf{r}_2$ if for every $g \in \mathbf{r}_2 \setminus \mathbf{r}_1$, there exists $f \in \mathbf{r}_1 \setminus \mathbf{r}_2$ such that $f \succ g$. Informally, $\mathbf{r}_1 <_{\mathbf{db}} \mathbf{r}_2$ states that $\mathbf{r}_1$ can be obtained from $\mathbf{r}_2$ by exchanging facts with more preferred facts. It can be verified that $<_{\mathbf{db}}$ is acyclic (by using that $\succ$ is acyclic). With this definition of $<_{\mathbf{db}}$, we obtain *g-repairs*: a consistent subset $\mathbf{r}$ of $\mathbf{db}$ is called a *globally optimal repair* (or *g-repair*) if there exists no consistent subset $\mathbf{r}'$ of $\mathbf{db}$ such that $\mathbf{r}' <_{\mathbf{db}} \mathbf{r}$.

Example 2. *Take the following database instance:*

R	1	2	
	a	b	(f_1)
	c	b	(f_2)
	c	d	(f_3)

Let $\Sigma = \{R : \{1\} \rightarrow \{2\}, R : \{2\} \rightarrow \{1\}\}$*, which expresses that neither column should contain duplicate values. The* $\oplus$*-repairs are:*

1	2
a	b
c	d

and

1	2
c	b

The left-hand relation is the only C-repair. If we assume $f_2 \succ f_1$ *and* $f_2 \succ f_3$*, then the right-hand relation is the only g-repair.*

4. COMPLEXITY OF CQA

We will restrict our attention to Boolean queries. Under this restriction, consistent query answering becomes a decision problem: given q, Σ, and $\mathbf{db}$, does q evaluate to *true* on every repair of $\mathbf{db}$? The input to this problem consists of three parts, and in the study of its complexity zero, one, or two parts can be fixed. In this paper, we take a *data complexity* perspective: we will fix both the query and the set of integrity constraints, and measure complexity with respect to the database instance. Alternative complexity analyses, which consider also queries and/or integrity constraints as part of the input, can be found in [4]. Thus, for every Boolean query q and set Σ of integrity constraints, we have the following problem:

$\mathsf{CERTAINTY}(q, \Sigma)$

INSTANCE: A database instance $\mathbf{db}$.

QUESTION: Does q evaluate to *true* on every repair of $\mathbf{db}$ with respect to Σ?

Of course, this definition is only meaningful when a repair notion has been established beforehand. We will write $\oplus$-$\mathsf{CERTAINTY}(q, \Sigma)$ if $\oplus$-repairs are intended. Furthermore, it is always understood that the database schema contains all relation names used in q or Σ.

We so far have defined consistent query answering for every individual pair q, Σ. It is also of interest to measure the complexity of consistent query answering for classes of queries and classes of integrity constraints. To this extent, let Q be a class of queries, and IC a class of integrity constraints. Let C be a complexity class.

- Consistent query answering for Q and IC is said to be *in* C if $\mathsf{CERTAINTY}(q, \Sigma)$ is in C for all $q \in \mathsf{Q}$ and $\Sigma \subseteq \mathsf{IC}$.
- Consistent query answering for Q and IC is said to be C-*complete* if it is in C and, moreover, $\mathsf{CERTAINTY}(q, \Sigma)$ is C-complete for some $q \in \mathsf{Q}$ and $\Sigma \subseteq \mathsf{IC}$.

For example, for symmetric-difference repairing, a correct claim is: *"Consistent query answering is* coNP*-complete for conjunctive queries and key dependencies."* Indeed, if q is a conjunctive query and Σ a set of key dependencies, then membership of $\oplus$-$\mathsf{CERTAINTY}(q, \Sigma)$ in coNP is straightforward: a succinct disqualification for a "no"-instance is any repair that falsifies q. coNP-completeness holds since it is known that $\oplus$-$\mathsf{CERTAINTY}(q_0, \Sigma_0)$ is coNP-hard for $q_0 = \exists x \exists y \exists z \, (R(x, y, z) \wedge S(z, x))$ and $\Sigma_0 = \{R : \{1, 2\} \rightarrow \{3\}, \, S : \{1\} \rightarrow \{2\}\}$. For a reason that will become apparent in Section 5.2, note that q_0 is self-join-free and that Σ_0 has only one key dependency per relation.

Arming et al. in [4] have carried out a thorough study on the complexity of consistent query answering for conjunctive queries and $\oplus$-repairs; data complexity results for different classes of integrity constraints are shown in Table 1. Triangles indicate whether the lower ($\vartriangle$) or the upper ($\triangledown$) complexity bound is of interest. For example, for coNP-completeness, $\triangledown$ and $\vartriangle$ denote, respectively, membership in coNP and coNP-hardness. If a line contains no reference to the literature, then its complexity result follows from other lines in the table.

Table 1 conveys an unpleasant message: consistent query answering is intractable already for very common queries and integrity constraints. However, some nuance is needed: coNP-completeness of consistent query answering for Q and IC tells us that the set $\{\mathsf{CERTAINTY}(q, \Sigma) \mid q \in \mathsf{Q}, \Sigma \subseteq \mathsf{IC}\}$ contains at least one intractable problem, but does not give us any hint about the boundary between tractable and intractable problems. It may still be the case that this set contains many tractable problems of practical interest. We will bring a more nuanced story in Section 5.

Descriptive Complexity of CQA.

Table 1 uses common *computational complexity* classes (P, coNP, Π_2^{P}). In the realm of (consistent) query answering, it may be more relevant to utilize *descriptive complexity*, which describes the complexity of the problem $\mathsf{CERTAINTY}(q, \Sigma)$ in some logic formalism, as explained next.

Notice first that every problem $\mathsf{CERTAINTY}(q, \Sigma)$ is actually a Boolean query (as introduced in the first paragraph of Section 2), mapping each database instance to either *true* or *false*. Let $\mathcal{L}$ be some logic language. We say that $\mathsf{CERTAINTY}(q, \Sigma)$ is

IC	$\oplus$-CERTAINTY(Σ, q)	Reference
FO	undecidable	
$\vee$-tgd	undecidable	
tgd	undecidable	[33, Theorem 7.2]
egds + weakly acyclic tgds	Π_2^{P}-complete $\triangledown$	[33, Theorem 6.3]
weakly acyclic tgds	Π_2^{P}-complete $\vartriangle$	[33, Theorem 6.3]
LAV tgd	in P	[33, Theorem 4.7]
IND	in P	
UC	Π_2^{P}-complete $\triangledown$ $\vartriangle$	[31, Lemma 4,Theorem 6]
full $\vee$-tgd	Π_2^{P}-complete $\vartriangle$	Modification of the $\vartriangle$ proof for UC [4]
full tgd	coNP-complete $\triangledown$ $\vartriangle$	[31][33, Theorem 5.5]
denial	coNP-complete $\triangledown$	[31]
egd	coNP-complete	
FD	coNP-complete	
key	coNP-complete $\vartriangle$	[12, Theorem 3.3]

Table 1: Data complexity results for consistent query answering, for conjunctive queries and $\oplus$-repairs.

expressible in $\mathcal{L}$ if there exists a formula Q in $\mathcal{L}$ such that the following are equivalent for every database instance **db**:

1. q is true on every repair of **db** with respect to Σ;
2. Q is true on **db**.

Such a formula Q, if it exists, is called a *consistent $\mathcal{L}$-rewriting of q* (with respect to Σ). The practice of constructing Q is often referred to as "query rewriting."

From a database perspective, the most attractive candidate for $\mathcal{L}$ is probably first-order predicate calculus, denoted FO. Indeed, if we can construct a consistent FO-rewriting of q with respect to Σ, then the problem CERTAINTY(q, Σ) can be solved by a single SQL query on existing RDBMS engines. Another good candidate for query rewriting is Datalog with stratified negation, whose data complexity is in P (and is complete for P). For the higher complexities in Table 1, more expressive logics are needed, such as variants of disjunctive Datalog [3, 19]. In general, by studying the descriptive complexity of problems CERTAINTY(q, Σ), we get a handle on the database languages that can be used to solve them.

Integrity Constraints from Different Classes.

In database applications, it is normal to have integrity constraints belonging to different classes, for example, a combination of inclusion and functional dependencies. Table 2 is somewhat unsatisfactory insofar as it does not consider unions of common classes of integrity constraints, except for unions of a set of egds and a weakly acyclic set of tgds. Note incidentally that in Fig. 1, the smallest class that includes both inclusion and functional dependencies is the class of all first-order logic constraints.

5. PRIMARY KEYS

The notion of primary key is fundamental and ubiquitous in relational database systems. Its study in the context of CQA has therefore attracted considerable attention and has revealed some interesting connections with other fields. This section shows that a nuanced complexity landscape hides behind the last line of Table 1, and discusses connections to CSP and probabilistic database systems.

5.1 Some Terminology

Recall that a key dependency is a functional dependency $R : X \rightarrow Y$ such that $X \cup Y$ contains every positive integer up to the arity of R. A database instance satisfies this key dependency if and only if it does not contain two distinct facts that agree on all positions in X. By a *set of primary keys*, we mean a set of key dependencies containing exactly one key dependency for each relation name in the database schema under consideration. If such a set of primary keys contains $R : X \rightarrow Y$, then X is said to be the *primary key* of R. For the sake of simplicity, it is commonly assumed that the primary key X of R is not empty and formed by the $|X|$ leftmost positions (i.e., $X = \{1, 2, \dots, |X|\}$ with $|X| \geq 1$).

A natural way for specifying repairs with respect to primary keys goes as follows. We say that two database facts are *key-equal* if they share the same relation name and agree on the primary key of their shared relation name. Given a database instance

db, the binary relation *"is key-equal to"* is obviously an equivalence relation on **db**; its equivalence classes are called the *blocks* of **db**. Obviously, **db** is consistent if and only if none of its blocks contains two or more database facts. Every *repair* of **db** is obtained by picking exactly one database fact from each of its blocks. The repairs defined in this way are both $\oplus$-repairs and C-repairs, and moreover are subset-repairs. In this section, we do not consider other conceivable ways of fixing primary key violations, such as replacing a duplicate primary key value with a fresh value.

The notion of block also occurs in probabilistic databases for modeling uncertainty: only one database fact of a block can be true, but we do not know which one holds true.

5.2 A Complexity Trichotomy

In Section 4, we explained what it means that consistent query answering is coNP-complete for conjunctive queries and primary keys. This is a rough result because it does not say anything about the boundary between tractable and intractable problems. The following trichotomy theorem from [21, 23] offers a more fine-grained complexity classification, albeit only for queries that are self-join-free.

THEOREM 1. *For every set Σ of primary keys and self-join-free Boolean conjunctive query q, the problem* $\mathsf{CERTAINTY}(q, \Sigma)$ *is in* FO, L*-complete, or* coNP*-complete. Moreover, it is decidable which of the three cases applies.*

The proof of Theorem 1 also yields a significant result in descriptive complexity, because it shows membership in L by expressing $\mathsf{CERTAINTY}(q, \Sigma)$ in symmetric stratified Datalog with some aggregation operator. Another promising observation in [23] is that tractability in L or FO obtains, roughly speaking, when all joins are foreign-to-primary key joins, which is undoubtedly the most common kind of join. In conclusion, for primary keys and self-join-free conjunctive queries, the most common cases of consistent query answering happen to be tractable, and all tractable cases can be solved by query rewriting in stratified Datalog with some aggregation operator.

For illustration, compare the following queries:

$$\begin{aligned} q_0 &= \exists x \exists y \exists z \left(S(\underline{z}, x) \wedge R(\underline{x, y}, z) \right), \\ q_1 &= \exists x \exists y \exists z \left(S(\underline{z}, x) \wedge T(\underline{x}, z) \right), \end{aligned}$$

where primary keys are underlined. It is known that consistent query answering is coNP-complete for q_0 (see Section 4), and in L (but not in FO) for q_1. This difference in complexity between q_0 and q_1 occurs because $S(\underline{z}, x)$ uses *all* primary key variables of the other atom in q_1, but does not so in q_0 (in particular, y does not occur in $S(\underline{z}, x)$). The join in q_0 is therefore not a foreign-to-primary key join.

An obvious open question is how to extend Theorem 1 to conjunctive queries with self-joins; we will have more to say about this in Section 5.3. A different partial extension of Theorem 1 appears in [22]: membership of $\mathsf{CERTAINTY}(q, \Sigma)$ in FO remains decidable if q is a self-join-free Boolean conjunctive query with clique-guarded negated atoms. Negation is called clique-guarded if whenever some variables x and y occur together in a negated atom, they also occur together in some non-negated atom.

For a reason that will become apparent in the next subsection, we state a P-coNP-complete dichotomy that immediately follows from Theorem 1:

COROLLARY 1. *For every set Σ of primary keys and self-join-free Boolean conjunctive query q, the problem* $\mathsf{CERTAINTY}(q, \Sigma)$ *is either in* P *or* coNP*-complete.*

5.3 Connections with CSP

A famous open conjecture is the following.

CONJECTURE 1. *For every set Σ of primary keys, for every query q that is a disjunction of Boolean conjunctive queries,* $\mathsf{CERTAINTY}(q, \Sigma)$ *is either in* P *or* coNP*-complete.*

Conjecture 1 generalizes Corollary 1 to unions of conjunctive queries, possibly with self-joins. The proof of Theorem 1 uses a predominantly logic approach, and it may be tempting to attack Conjecture 1 by the same logic apparatus. However, for reasons explained in the next paragraph, such a line of attack is unlikely to lead to success.

In her article with the questioning title *"Why Is It Hard to Obtain a Dichotomy for Consistent Query Answering?"* [18], Fontaine establishes connections between computational complexities in CQA and constraint satisfaction problems (CSPs). These connections were further investigated in [26]. One of Fontaine's findings is that a proof of Conjecture 1 would imply Bulatov's dichotomy theorem for conservative CSPs. The three published proofs [5, 7, 8] of the latter theorem use an algebraic approach developed over many years. Therefore, it is hardly conceivable that Conjecture 1 can be settled by the logic approach developed for Theorem 1. A different way to attack Conjecture 1 would be to show that it is implied by the recently proved CSP dichotomy theorem [9, 35].

5.4 Counts and Probabilities

For a set Σ of primary keys, the number of repairs of a given database instance is finite and can easily be calculated. Rather than asking whether all repairs satisfy a Boolean query q, a computationally more difficult problem is to determine how many repairs satisfy q. It has been shown [28] that for every set Σ of primary keys and self-join-free Boolean conjunctive query q, the following problem is either in FP or $\sharp\mathsf{P}$-complete: given a database instance **db**, determine the number of repairs of **db** that satisfy q. It is an open conjecture that this dichotomy result remains true over the class of all Boolean conjunctive queries. When all primary keys are singletons, this conjecture has been shown to be true [29, 34]. In these results, $\sharp\mathsf{P}$-hardness is with respect to polynomial-time Turing reductions; Calautti et al. [10] have recently studied the consequences of using many-one logspace reductions instead, which are a weaker form of reduction.

We close this section by pointing out an intimate relationship between the counting variant of $\mathsf{CERTAINTY}(q, \Sigma)$ and query answering in probabilistic databases. Assume a probability distribution over the set of all repairs of some database instance **db**. The probability of a Boolean query q, denoted $Pr\,(q)$, is then defined to be the sum of the probabilities of the repairs that satisfy the query. A common assumption is that facts of distinct blocks are independent, i.e., if $A_1, A_2, \dots, A_k$ are facts belonging to k distinct blocks, then $Pr\left(\bigwedge_{i=1}^{k} A_i\right) = \prod_{i=1}^{k} Pr\,(A_i)$. Notice here that (conjunctions of) atomic facts are Boolean queries, for which we have defined Pr. If this independence assumption holds true, then the probability distribution over the set of all repairs is fully determined if we know the marginal probability $Pr\,(A)$ for each fact A in **db**. These probabilities can then be listed as illustrated in the following table; it is also common to underline primary keys, and to separate blocks by dashed lines.

S	<u>S#</u>	SNAME	STATUS	CITY	Pr
	S1	Smith	20	London	1
	S2	Jones	10	Paris	0.7
	S2	Jones	15	Paris	0.3

Then, for a fixed Boolean query q, there is a natural shift from counting to calculating probabilities: given a database instance and the marginal probabilities of its facts, determine $Pr\,(q)$.

The probabilistic data model just described is almost the same as the *block-independent disjoint* (BID) probabilistic data model [13]. The only difference is that in BID probabilistic database instances, the marginal probabilities of the facts of a same block need not sum up to one. This difference emerges because *possible worlds* in the BID probabilistic data model are merely restricted not to contain two distinct facts from a same block, which leaves the possibility of selecting no fact from a block. Repairs, on the other hand, must contain exactly one fact from each block. A complexity dichotomy similar to the one previously cited holds: for every self-join-free Boolean conjunctive query q, the data complexity of evaluating q on BID probabilistic database instances is either in FP or $\sharp\mathsf{P}$-hard [14].

6. REPAIR CHECKING

Assume a repair notion has been fixed, for example, symmetric-difference repairing. Repair checking, then, is the following decision problem: given a set Σ of integrity constraints and two database instances **db** and **r**, determine whether **r** is a repair of **db**. Since this paper's focus is on data complexity, we define this problem for any fixed set Σ:

$\mathsf{REPAIR\text{-}CHECKING}(\Sigma)$

> INSTANCE: Database instances **db** and **r**.
>
> QUESTION: Is **r** a repair of **db** with respect to Σ?

We will write $\oplus$-$\mathsf{REPAIR\text{-}CHECKING}(\Sigma)$ when we assume $\oplus$-repairs.

Repair checking is relevant to (the complexity of) consistent query answering: a method for solving the complement of $\mathsf{CERTAINTY}(q, \Sigma)$, with **db** as input, consists in non-deterministically guessing a database instance **r**, and checking whether **r** falsifies q and whether the pair **db**, **r** is a "yes"-instance of $\mathsf{REPAIR\text{-}CHECKING}(\Sigma)$. This method is effective when the size of **r** can be polynomially bounded in the size of **db**, as is the case for $\oplus$-repairs with respect to denial constraints and full tgds.

The complexity of repair checking for a class IC of integrity constraints can be expressed in the following terms, where C denotes a complexity class:

- Repair checking for IC is said to be *in* C if $\mathsf{REPAIR\text{-}CHECKING}(\Sigma)$ is in C for all $\Sigma \subseteq \mathsf{IC}$.
- Repair checking for IC is said to be C-*complete* if it is in C and, moreover, for some $\Sigma \subseteq \mathsf{IC}$, $\mathsf{REPAIR\text{-}CHECKING}(\Sigma)$ is C-complete.

Arming et al. in [4] have carried out a thorough study on the complexity of $\oplus$-repair checking; data complexity results are shown in Table 2. Results on C-repair checking appear in [1, 25]. A similar caveat

IC	$\oplus$-REPAIR-CHECKING(Σ)	Reference
FO	coNP-complete $\triangledown$	[1, Proposition 4]
$\vee$-tgd	coNP-complete	
tgd	coNP-complete	
weakly acyclic tgds	coNP-complete $\vartriangle$	[1, Theorem 7]
LAV tgd	in P	[33, Theorem 4.9]
weakly acyclic LAV tgds	in L	[1, Theorem 3]
IND	in P	
UC	coNP-complete $\triangledown$ $\vartriangle$	[31, Lemma 4, Corollary 3]
full $\vee$-tgd	coNP-complete $\vartriangle$	Modification of the $\vartriangle$ proof for UC [4]
full tgd	P-complete $\triangledown$ $\vartriangle$	[30, Theorem 3.7][1, Theorem 5]
denial	in L	[1, Proposition 5]
egd	in L	
FD	in L	
key	in L	

Table 2: Data complexity results for $\oplus$-repair checking.

as in Section 4 is in order here: coNP-completeness of repair checking for IC tells us nothing about the boundary between tractable and intractable problems in the set $\{\text{REPAIR-CHECKING}(\Sigma) \mid \Sigma \subseteq \text{IC}\}$.

For g-repairs (see Section 3 for the definition of g-repairs), it follows from Proposition 5 and Theorem 2 in [32] that g-repair checking is coNP-complete for functional dependencies. Note that in the case of g-repairs, the input to REPAIR-CHECKING(Σ) also contains the binary preference relation $\succ$ on $\mathbf{db}$. A more fine-grained complexity classification for g-repair checking appears in [16], where it is shown that for every set Σ of functional dependencies, the problem REPAIR-CHECKING(Σ) is either in P or coNP-complete, and it is decidable which of the two cases applies. More complexity results on variants of g-repair checking appear in [20].

Repair Counting.

Livshits and Kimelfeld in [24] study the problem of counting database repairs. They show, among others, that for every set of functional dependencies, the data complexity of the following problem is either in FP or $\sharp$P-complete: given a database instance $\mathbf{db}$, determine the number of $\oplus$-repairs of $\mathbf{db}$. Remind here that for functional dependencies, all $\oplus$-repairs are subset-repairs.

Integrity Constraints from Different Classes.

Few complexity studies in CQA consider combining integrity constraints from different classes, which is nevertheless most common in practice. The following two results show that such combinations can entail an increase in the data complexity of repair checking. First, from Theorem 4.6 in [12], it follows that $\oplus$-repair checking for INDs and FDs taken together is coNP-complete. This is to be contrasted with the tractable complexities for IND and FD in Table 2. Second, by [1, Theorem 8], there are a weakly acyclic set Σ_1 of LAV tgds and a set Σ_2 of egds such that $\oplus$-REPAIR-CHECKING($\Sigma_1 \cup \Sigma_2$) is coNP-complete, whereas Table 2 shows that for $i \in \{1, 2\}$, $\oplus$-REPAIR-CHECKING(Σ_i) is tractable.

7. CONCLUDING THOUGHTS

Consistent query answering has been studied for all common classes of integrity constraints. Tables 1 and 2 show some of the rich body of theoretical results obtained over the past twenty years. These results, however, are still open to refinement. For example, it has been known since the early years that consistent query answering is coNP-complete for conjunctive queries and key dependencies, as reported by the last line of Table 1. But still today it remains an open conjecture that for every Boolean conjunctive query q and set Σ of key dependencies, the problem CERTAINTY(q, Σ) can be classified as either coNP-complete or in P. It took until recently to show this conjecture under the serious restrictions of self-join-free queries and primary keys. For more expressive queries and integrity constraints, such detailed complexity classifications are missing.

We conclude with a reflection on the notion of repairing. In Section 3.2, we have modeled database repairing by an acyclic binary relation $\leq_{\mathbf{db}}$ on the set of consistent database instances, where the intuition behind $\mathbf{r}_1 \leq_{\mathbf{db}} \mathbf{r}_2$ is that $\mathbf{r}_1$ is at least as close to $\mathbf{db}$ as $\mathbf{r}_2$. The repairs, then, are the consistent database instances that are most close to $\mathbf{db}$. The relation $\leq_{\mathbf{db}}$ is never explicitly given, but rather implicitly specified by using some embodiment of the principle of minimal change, nearly always in a manner that is agnostic of the meaning of the data. We believe that it would be worthwhile to explore capabilities for further restricting "the legal" repairs—in the same way as integrity constraints restrict "the legal" databases. Such a capability is already provided, for example, by the preference relation on database facts in prioritized repairs (see Section 3). But this preference relation, rather than being given explicitly, could be specified implicitly in some declarative fashion, capturing more of the meaning of the data. Proposals for such formalism can be found in [11, 27].

8. REFERENCES

[1] F. N. Afrati and P. G. Kolaitis. Repair checking in inconsistent databases: Algorithms and complexity. In *ICDT*, pages 31–41, 2009.

[2] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[3] M. Arenas, L. E. Bertossi, and J. Chomicki. Answer sets for consistent query answering in inconsistent databases. *TPLP*, 3(4-5):393–424, 2003.

[4] S. Arming, R. Pichler, and E. Sallinger. Complexity of repair checking and consistent query answering. In *ICDT*, pages 21:1–21:18, 2016.

[5] L. Barto. The dichotomy for conservative constraint satisfaction problems revisited. In *LICS*, pages 301–310, 2011.

[6] L. E. Bertossi. Database repairs and consistent query answering: Origins and further developments. In *PODS*, pages 48–58, 2019.

[7] A. A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Log.*, 12(4):24:1–24:66, 2011.

[8] A. A. Bulatov. Conservative constraint satisfaction re-revisited. *J. Comput. Syst. Sci.*, 82(2):347–356, 2016.

[9] A. A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *FOCS*, pages 319–330, 2017.

[10] M. Calautti, M. Console, and A. Pieris. Counting database repairs under primary keys revisited. In *PODS*, pages 104–118, 2019.

[11] L. Caroprese, S. Greco, and E. Zumpano. Active integrity constraints for database consistency maintenance. *IEEE Trans. Knowl. Data Eng.*, 21(7):1042–1058, 2009.

[12] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

[13] N. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: Diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.

[14] N. N. Dalvi, C. Ré, and D. Suciu. Queries and materialized views on probabilistic databases. *J. Comput. Syst. Sci.*, 77(3):473–490, 2011.

[15] C. J. Date. *An introduction to database systems (7. ed.)*. Addison-Wesley-Longman, 2000.

[16] R. Fagin, B. Kimelfeld, and P. G. Kolaitis. Dichotomies in the complexity of preferred repairs. In *PODS*, pages 3–15, 2015.

[17] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[18] G. Fontaine. Why is it hard to obtain a dichotomy for consistent query answering? *ACM Trans. Comput. Log.*, 16(1):7:1–7:24, 2015.

[19] G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Trans. Knowl. Data Eng.*, 15(6):1389–1408, 2003.

[20] B. Kimelfeld, E. Livshits, and L. Peterfreund. Detecting ambiguity in prioritized database repairing. In *ICDT*, pages 17:1–17:20, 2017.

[21] P. Koutris and J. Wijsen. Consistent query answering for self-join-free conjunctive queries under primary key constraints. *ACM Trans. Database Syst.*, 42(2):9:1–9:45, 2017.

[22] P. Koutris and J. Wijsen. Consistent query answering for primary keys and conjunctive queries with negated atoms. In *PODS*, pages 209–224, 2018.

[23] P. Koutris and J. Wijsen. Consistent query answering for primary keys in logspace. In *ICDT*, pages 23:1–23:19, 2019.

[24] E. Livshits and B. Kimelfeld. Counting and enumerating (preferred) database repairs. In *PODS*, pages 289–301, 2017.

[25] A. Lopatenko and L. E. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *ICDT*, pages 179–193, 2007.

[26] C. Lutz and F. Wolter. On the relationship between consistent query answering and constraint satisfaction problems. In *ICDT*, pages 363–379, 2015.

[27] M. V. Martinez, F. Parisi, A. Pugliese, G. I. Simari, and V. S. Subrahmanian. Policy-based

inconsistency management in relational databases. *Int. J. Approx. Reasoning*, 55(2):501–528, 2014.

[28] D. Maslowski and J. Wijsen. A dichotomy in the complexity of counting database repairs. *J. Comput. Syst. Sci.*, 79(6):958–983, 2013.

[29] D. Maslowski and J. Wijsen. Counting database repairs that satisfy conjunctive queries with self-joins. In *ICDT*, pages 155–164, 2014.

[30] S. Staworko. *Declarative Inconsistency Handling in Relational and Semi-Structured Databases*. PhD thesis, State University of New York at Buffalo, 2007.

[31] S. Staworko and J. Chomicki. Consistent query answers in the presence of universal constraints. *Inf. Syst.*, 35(1):1–22, 2010.

[32] S. Staworko, J. Chomicki, and J. Marcinkowski. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.*, 64(2-3):209–246, 2012.

[33] B. ten Cate, G. Fontaine, and P. G. Kolaitis. On the data complexity of consistent query answering. *Theory Comput. Syst.*, 57(4):843–891, 2015.

[34] J. Wijsen. Corrigendum to "Counting database repairs that satisfy conjunctive queries with self-joins". *CoRR*, abs/1903.12469, 2019.

[35] D. Zhuk. A proof of CSP dichotomy conjecture. In *FOCS*, pages 331–342, 2017.