An Operational Approach to Consistent Query Answering

Marco Calautti
School of Informatics
University of Edinburgh, UK
mcalautt@inf.ed.ac.uk

Leonid Libkin
School of Informatics
University of Edinburgh, UK
libkin@inf.ed.ac.uk

Andreas Pieris
School of Informatics
University of Edinburgh, UK
apieris@inf.ed.ac.uk

ABSTRACT

Consistent query answering (CQA) aims to find meaningful answers to queries when databases are inconsistent, i.e., do not conform to their specifications. Such answers must be certainly true in all repairs, which are consistent databases whose difference from the inconsistent one is minimal, according to some measure. This task is often computationally intractable, and much of CQA research concentrated on finding islands of tractability. Nevertheless, there are many relevant queries for which no efficient solutions exist, which is reflected by the limited practical applicability of the CQA approach. To remedy this, one needs to devise a new CQA framework that provides explicit guarantees on the quality of query answers. However, the standard notions of repair and certain answers are too coarse to permit more elaborate schemes of query answering. Our goal is to provide a new framework for CQA based on revised definitions of repairs and query answering that opens up the possibility of efficient approximate query answering with explicit guarantees. The key idea is to replace the current declarative definition of a repair with an *operational* one, which explains *how* a repair is constructed, and how likely it is that a consistent instance is a repair. This allows us to define how certain we are that a tuple should be in the answer. Using this approach, we study the complexity of both exact and approximate CQA. Even though some of the problems remain hard, for many common classes of constraints we can provide meaningful answers in reasonable time, for queries going far beyond the standard CQA approach.

ACM Reference Format:
Marco Calautti, Leonid Libkin, and Andreas Pieris. 2018. An Operational Approach to Consistent Query Answering. 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/10.1145/3196959.3196966

1 INTRODUCTION

Consistent query answering (CQA) is an elegant idea introduced in the late 1990s by Arenas, Bertossi, and Chomicki [2] that has been extensively studied since. The main premise is that databases are often *inconsistent*, i.e., do not conform to their specifications in the form of integrity constraints. The reason behind this is that data is not perfect and clean: it may come, for example, from several conflicting sources. Data cleaning attempts to fix this problem, but it is not always possible. In such a case, CQA aims to deliver meaningful answers to queries that can still be obtained from inconsistent data. The key elements of the CQA approach are [2, 5]:

(1) the notion of a *repair* of an inconsistent database D, i.e., a consistent database D' whose difference with D is minimal;
(2) the notion of query answering based on *certain answers*, i.e., one looks at answers that are true in all repairs.

Since there could be many repairs, finding certain answers is most commonly in coNP in data complexity, and it is very often coNP-hard, even for conjunctive queries [12, 40]. This led to a large body of work on drawing the tractability boundary for query answering [20–22, 28, 29]. Much of it views a dichotomy result, classifying all query answering into tractable and coNP-hard, as the ultimate goal of the CQA endeavor, although there are strong indications that for some common classes of database integrity constraints obtaining such a dichotomy would be extremely hard [19, 35].

Nevertheless, even obtaining good sufficient conditions for tractability leaves many relevant queries beyond reach of the CQA approach. Thus, the standard approach, while yielding good theoretical results, appears to be a bit of dead end which is reflected by its limited practical applicability [20, 31]. We would like to rectify this. We believe that the ultimate goal of a practically applicable CQA approach should be *efficient approximate query answering* with explicitly stated guarantees. However, in the current state of affairs this goal does not seem to be attainable. Efficient probabilistic algorithms with bounded one-sided or two-sided error are unlikely for CQA: placing it in tractable randomized complexity classes such as RP or BPP would imply that the polynomial hierarchy collapses [26]. For coming up with more refined approximation techniques, the current CQA framework lacks flexibility and finer details related to its key concepts, as we now explain.

The standard approach simply imposes a condition that a repair and an inconsistent database must satisfy, and then defines query answering by means of certain answers. This provides little information for two reasons. First, the notion of repairs does not explain how repairs are constructed; we know that an instance is a repair or not. Moreover, we do not know when, and crucially why, is one repair more likely to appear than another. Second, the notion of certain answers only says that a tuple is entailed by all repairs, or is not entailed by some repair. But the former is too strict and the latter not very useful. Instead, we would like to know how likely a tuple is to be in the answer. However, with few exceptions [23], the standard CQA approach does not consider this. Thus, we strongly believe that making progress towards turning CQA into a practically viable approach requires a different framework for the key notions of repairs and query answering.

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

ACM ISBN 978-1-4503-4706-8/18/06…$15.00
https://doi.org/10.1145/3196959.3196966

Our main idea is to replace the declarative approach to repairs with an *operational* one that explains the process of constructing a repair. As it gives us a finer understanding of why an instance is a repair, it also leads to more refined ways of answering queries by letting us define precisely *how certain* we are that a tuple should be in the answer. This approach offers us a lot more flexibility: for example, in defining the operational approach we can also reason about more or less likely updates that lead to a repair. This in turn lets us define probabilistic guarantees for query answers, and construct efficient algorithms that produce such guarantees for common classes of constraints that have previously been beyond reach of the CQA approach.

Outline of the operational framework. The key elements of the new approach to database repairs are:

(1) the notion of violations of constraints;
(2) repairing sequences of operations on databases;
(3) assigning likelihood to repairs based on operations used in repairing sequences; and
(4) flexible query answering based on the likelihood of a tuple appearing in an answer on different repairs.

A repairing sequence applies *operations* to a database to eliminate violations of constraints, and does so until a consistent database is produced. For example, for relational databases, one may consider sequences of operations inserting and deleting tuples. For a database D and a set Σ of constraints, a *violation* of a constraint $\kappa \in \Sigma$ explains why $D \not\models \kappa$. Constraints are usually logical statements of the form $\forall \bar{x}\ \alpha(\bar{x}) \rightarrow \beta(\bar{x})$. Then a violation is given by a tuple $\bar{a}$ such that $D \models \alpha(\bar{a})$ but $D \not\models \beta(\bar{a})$. These are the violations we deal with here, but in general there could be others, for other types of data and/or constraints. We write $\mathsf{V}(D, \Sigma)$ for the set of all violations of Σ in D.

We now formulate a minimal set of requirements for repairing sequences: these ought to be true in every data model and for every notion of constraints. A *repairing sequence* is a sequence of operations $s = op_1, op_2, \ldots, op_n$. Starting with a database D, these produce a sequence of databases $D_0 = D$, $D_1 = op_1(D_0)$, …, $D_i = op_i(\ldots op_1(D)\ldots)$, etc. The minimal requirements imposed on such repairing sequences are:

(1) each operation must be *justified*, i.e., remove at least one violation; and
(2) a violation eliminated by op_i cannot be reintroduced later by an operation op_j for $j > i$.

Formally, these are stated as follows:

req1 $\mathsf{V}(D_{i-1}, \Sigma) - \mathsf{V}(op_i(D_{i-1}), \Sigma) \neq \varnothing$ for every $i > 0$.
req2 $(\mathsf{V}(D_{i-1}, \Sigma) - \mathsf{V}(D_i, \Sigma)) \cap \mathsf{V}(D_j, \Sigma) = \varnothing$ for $j > i$.

In a specific model of data and updates, these minimal requirements may be supplemented by others. For instance, for relational databases and insertion/deletion updates, the same tuple should not belong to both an insertion and a deletion operation of a repairing sequence: it makes no sense to insert a tuple only to delete it later.

For a sequence $s = op_1, \ldots, op_n$ of operations, its result $s(D)$ is $op_n(\ldots op_1(D)\ldots)$. A repairing sequence attempts to construct a repair: it may succeed, if $s(D)$ satisfies Σ, or it may fail, if no further operations can be applied and $s(D)$ still violates Σ. We view databases $s(D)$ when s succeeds as our *operational repairs*.

There is no a priori reason to believe that all such repairs are equally likely. The idea that not all repairs are created equal has already been explored in the CQA context [18, 36], but here we take it further. The operational approach lets us say *why* one repair is more likely, and also assign a *quantitative measure* to such output. For this, all we need is a measure of likelihood of operations in a given state of repair. Say we reached some state $s(D)$ by applying a sequence $s = op_1, \ldots, op_n$ of operations to D. If $s(D)$ is not yet consistent, there may be several operations $op'_1, \ldots, op'_k$ one can apply to remove violations. We can assign the notion of likelihood to those, which in turn leads to the notion of likelihood of a repair.

This idea is very naturally modeled by tree-shaped *Markov chains* [27]. If we assign probabilities $p_1, \ldots, p_k$ to operations $op'_1, \ldots, op'_k$ that can follow the sequence s so that p_i's add up to 1, we have a Markov chain on repairing sequences. Then the probability of a particular successful repairing sequence arising in such a Markov chain is given by what is known as its hitting distribution. This is how we define the likelihood of a particular repair.

Probabilities can be naturally assigned to repairing operations in many scenarios leading to inconsistencies. Consider, for instance, a data integration scenario that results in an integrated database containing facts $R(a, b)$ and $R(a, c)$ that violate the constraint that the first attribute of R is a key. Suppose we have a level of trust in each of the sources supplying data; say we believe that each is 50% reliable. Then with probability $0.5 \cdot 0.5 = 0.25$ we do not trust either tuple and apply the operation that removes both facts. With probability $(1 - 0.25)/2 = 0.375$ we remove either $R(a, b)$ or $R(a, c)$. The standard CQA aproach [2] only allows the removal of one of the two facts (with equal probability 0.5 in this case). It somehow assumes that we trust at least one of the sources, even though we know they are in conflict. Our approach is more flexible, as we account (with a smaller probability) for the case when neither source supplies correct facts. We shall expand this in Section 3.

The notion of operational repair fulfills our goal of providing new flexible ways of query answering: for a given tuple $\bar{t}$, we can add up the likelihoods of all repairing sequences s for which $\bar{t} \in Q(s(D))$ to get the likelihood that $\bar{t}$ is in the answer to Q. This is a much more refined notion than the usual certain answers. It allows us to devise approximation schemes with different types of guarantees for answering not only conjunctive, but arbitrary first-order queries in the CQA framework. In fact, as an application of our framework we shall study the complexity of different types of approximation schemes. While some remain computationally hard, we show that for a large class of updates, efficient approximations with probabilistic guarantees exist for all first-order queries and most common database constraints.

Outline of main results. Our main contributions include the new operational framework for database repairs and consistent query answering, and the study of its exact and approximate complexity.

The operational approach, described in detail in Section 3, formalizes the notions informally presented earlier: constraints and their violations, justified operations, repairing sequences, and Markov chains on such repairing sequences that let us compare their relative importance. This culminates in the definition of an operational repair and a new semantics of query answering based on the degree of certainty that a tuple is in the answer. Since operational

repairs have probabilities assigned to them, this degree of certainty is formally defined as a conditional probability that a tuple is in the query answer, under the condition that the database on which the query is asked is an operational repair. We also present detailed examples to illustrate the new framework.

With the framework in place, we study the complexity of query answering. Since operational repairs can encode the classical repairs of [2], it is not surprising that the complexity of query answering for first-order (and even conjunctive) queries is intractable. We pinpoint the exact complexity: $\mathrm{FP}^{\#\mathrm{P}}$-complete (note that the output of the query answering in this case is not yes or no, but rather a number, namely the conditional probability, as explained above).

With this bound, one looks for approximations, and given the probabilistic nature of query answers, we look for approximations via randomized algorithms. There are two types of guarantees for calculating the conditional probability p of a tuple $\bar{t}$ by means of a randomized algorithm that returns a number a. Either $|a - p| \leq \epsilon \cdot p$, for a fixed $\epsilon > 0$ (multiplicative error guarantees), or $|a - p| \leq \epsilon$ (additive error guarantees). Since a is the output of a randomized algorithm, we require these to hold with a high probability, say at least $1-\delta$ for small $1 > \delta > 0$. Multiplicative error algorithms (so-called FPRAS: fully polynomial-time randomized approximation scheme) are more common in the literature since the *relative* error between the output of an FPRAS and the value we want to approximate is bounded by ϵ. In the case of additive error algorithms, only the *absolute* error is bounded by ϵ, whereas the relative error increases as the value we want to approximate decreases. Nevertheless, additive error algorithms are equally useful for our purposes since we are approximating probabilities. Thus, having a high relative error for tuples with small probability is a reasonable price to pay, since such tuples are much less important than tuples with high probability.

We establish two results: our query answering problem does not admit an FPRAS (under some widely believed complexity-theoretic assumptions) but it does admit a polynomial-time randomized approximation algorithm with additive error guarantees. The latter happens under a mild restriction on the Markov chain that it does not admit failing sequences of updates, i.e., sequences that cannot be extended but do not yet repair the database. This is a common occurrence and it covers such common cases as key (or, more generally, EGD) violations. We describe this randomized algorithm, and discuss how it can be implemented in practice.

Organization. Notations are given in Section 2. In Section 3 we present the basics of the operational approach, and in Section 4 we describe query answering under this approach and study its complexity. In Section 5 we study approximations of the query answering problem with different types of guarantees. In Section 6 we give an extensive list of open problems that we hope to solve with the new framework.

2 PRELIMINARIES

In this section, we recall the basics on relational databases and constraints, and we fix some basic notation.

Relational Databases. We assume a countably infinite set $\mathbf{C}$ of *constants* from which database elements are drawn. A *(relational) schema* $\mathbf{S}$ is a finite set of relation symbols (or predicates) with associated arity. We write R/n to denote that R has arity n. A *fact* over $\mathbf{S}$ is an expression of the form $R(c_1, \ldots, c_n)$, where $R/n \in \mathbf{S}$ with $n > 0$, and $c_i \in \mathbf{C}$ for each $i \leq n$. A *database instance* (or simply *database*) over $\mathbf{S}$ is a finite set of facts over $\mathbf{S}$. The *active domain* of a database D, denoted $dom(D)$, is the set of constants in D.

Constraints. We shall be using standard database constraints: tuple- and equality-generating dependencies as well as denial constraints. Let $\mathbf{V}$ be a countably infinite set of *variables* disjoint from $\mathbf{C}$. An *atom* over a schema $\mathbf{S}$ is an expression $R(t_1, \ldots, t_n)$, where $R/n \in \mathbf{S}$, and $t_i \in \mathbf{C} \cup \mathbf{V}$ for each $1 \leq i \leq n$ (thus, a fact is an atom without variables). Our constraints over a schema $\mathbf{S}$ are defined as first-order sentences:

- a *tuple-generating dependency (TGD)* is of the form
$$\forall \bar{x} \forall \bar{y} \big(\varphi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})\big), \tag{1}$$
where φ and ψ are non-empty conjunctions of atoms;
- an *equality-generating dependency (EGD)* is of the form
$$\forall \bar{x} \big(\varphi(\bar{x}) \rightarrow x_i = x_j\big), \tag{2}$$
where φ is a non-empty conjunction of atoms, and x_i, x_j are variables of $\bar{x}$;
- a *denial constraint (DC)* is of the form
$$\forall \bar{x}\, \neg\varphi(\bar{x}), \tag{3}$$
where φ is a non-empty conjunction of atoms.

A database D satisfies a constraint if the corresponding sentence is true in D. In what follows, we use commas instead of $\wedge$ for conjunctions of atoms, as is common in the rule-based syntax of conjunctive queries. Recall that EGDs can express keys and functional dependencies; for instance, to say that the first attribute of R is a key, we write $\forall x, y, z\, (R(x, y), R(x, z) \rightarrow y = z)$. TGDs can express inclusion dependencies, e.g., $\forall x, y\, (R(x, y) \rightarrow \exists z\, S(z, x))$ is the inclusion dependency $R[1] \subseteq S[2]$. Thus the combination of EGDs and TGDs can express foreign keys. With denial constraints, one can express conditions such as disjointness of attribute domains: e.g., $\forall x, y, z\, \neg(R(x, y), R(z, x))$ says that the same constant cannot be a value of both the first and the second attribute of R. As is common, we omit universal quantifiers from TGDs, EGDs, and DCs; that is, every variable that occurs in them, except those existentially quantified in TGDs, is assumed to be universally quantified.

It is convenient to view satisfaction of these constraints via *homomorphisms*; this will be useful for formalizing the notion of violations. For a set of atoms A, we let $dom(A)$ be the set of all constants and variables occurring in A. A *homomorphism* from a set of atoms A to a set of atoms A' is a mapping $h : dom(A) \rightarrow dom(A')$, which is the identity on $\mathbf{C}$, such that, for every atom $R(\bar{t}) \in A$, the atom $R(h(\bar{t}))$ is in A'. Let $h(A) = \{R(h(\bar{t})) \mid R(\bar{t}) \in A\}$.

Notice that a conjunction of atoms in a formula can be viewed as a set of atoms (i.e., the tableau). Thus we can talk about homomorphisms from a conjunction of atoms to a database. In view of this, the notion of satisfaction of constraints can be restated as follows. A database D satisfies

- TGD (1) if for every homomorphism h from $\varphi(\bar{x}, \bar{y})$ to D, there is a homomorphism h' from $\psi(h(\bar{x}), \bar{z})$ to D;
- EGD (2) if, for every homomorphism h from $\varphi(\bar{x})$ to D, we have $h(x_i) = h(x_j)$;
- DC (3) if there is *no* homomorphism from $\varphi(\bar{x})$ to D.

Henceforth, we use the term *constraint* to refer to TGDs, EGDs and DCs. A database D is *consistent* with a set Σ of constraints, written $D \models \Sigma$, if D satisfies each constraint of Σ; otherwise, D is *inconsistent* with Σ.

Queries. We consider *first-order queries* $Q(\bar{x})$ that are expressions of the form $\{\bar{x} \mid \varphi\}$, where φ is a first-order formula with free variables $\bar{x}$. The output of Q on a database D is the set of tuples $Q(D) = \{\bar{c} \in dom(D)^{|\bar{x}|} \mid D \models \varphi(\bar{c})\}$.

Consistent Query Answering. As said in Section 1, inconsistent databases are a real-life phenomenon that arise due to many reasons such as integration of conflicting sources. To obtain meaningful answers from inconsistent data, [2] introduced the notion of consistent query answers, which we now recall.

For two databases D, D', the measure of distance between them is defined as their symmetric difference $\Delta(D, D') = (D - D') \cup (D' - D)$. Then for an inconsistent database D (w.r.t. a set of constraints Σ), a consistent database D' over $dom(D)$ and constants used in Σ is a *repair* if $\Delta(D, D')$ is minimal w.r.t. to subset inclusion. In other words, there is no other such consistent database D'' for which $\Delta(D, D'') \subsetneq \Delta(D, D')$. We denote the set of repairs of D w.r.t Σ by $[\![D]\!]^{\mathsf{ABC}}_{\Sigma}$; i.e., the Arenas-Bertossi-Chomicki (ABC) semantics of an inconsistent database. Consistent query answers, as defined in [2], are certain answers under the ABC semantics. Given a database D, a set of constraints Σ, and a query Q, the *consistent answer* to Q w.r.t. D and Σ, is the set of tuples $\bigcap\{Q(D') \mid D' \in [\![D]\!]^{\mathsf{ABC}}_{\Sigma}\}$.

3 THE OPERATIONAL APPROACH

We propose a new operational approach to CQA, based on the concepts of operations, violations of constraints and repairing sequences, and ways to assign likelihood to such sequences. We proceed to define these concepts for relational databases and constraints from the three classes seen earlier (TGDs, EGDs, and DCs) in a way that satisfy the minimal requirements from Section 1.

Operations and Violations

The notion of operation is the building block of our approach. For relational databases, the operations that we consider are standard updates $+F$ that add a set of facts F using constants occurring in the database and the constraints, or $-F$ that remove F from a database. To formalize this, we define the *base*, denoted by $\mathsf{B}(D, \Sigma)$, for a database D and a set Σ of constraints over a schema $\mathbf{S}$, as the set of all facts $R(c_1 \ldots, c_n)$ where $R/n \in \mathbf{S}$, and $c_1, \ldots, c_n$ are constants that occur in $dom(D)$ or in Σ. We use the notation $\mathcal{P}(\cdot)$ for powerset.

Definition 3.1. (**Operation**) For a database D and a set Σ of constraints, a (D, Σ)-*operation* is a function $op : \mathcal{P}(\mathsf{B}(D, \Sigma)) \to \mathcal{P}(\mathsf{B}(D, \Sigma))$ such that either:

(1) $op(D') = D' \cup F$, for every $D' \in \mathcal{P}(\mathsf{B}(D, \Sigma))$; or
(2) $op(D') = D' - F$, for every $D' \in \mathcal{P}(\mathsf{B}(D, \Sigma))$

for a set of facts $F \subseteq \mathsf{B}(D, \Sigma)$. We shall refer to these operations as $+F$ or $-F$ respectively. ∎

Technically speaking, the operations $+F$ and $-F$ depend on D and Σ, as they are only defined over $\mathsf{B}(D, \Sigma)$. Since D and Σ will be clear from the context, we may refer to them simply as operations, omitting D and Σ. Also when F contains a single atom $R(\bar{a})$, we write $+R(\bar{a})$ and $-R(\bar{a})$ instead of the more formal $+\{R(\bar{a})\}$ and $-\{R(\bar{a})\}$. At this point, let us clarify that, although we focus on insertions and deletions of facts as described above, the proposed framework can be extended by considering also other forms of operations such as insertions with null values [6, 7] or attribute-based operations [39]. Actually, as discussed in Section 6, these are issues that we are planning to address in the near future. The idea is then to iteratively apply operations, starting from a database D, until we reach a database that is consistent with Σ. However, by itself this does not meet the minimal requirement **req1** that at least one violation should be resolved. To impose this requirement, we need to keep track of all the reasons that cause the inconsistency of D with Σ. This brings us to the notion of constraint violation.

Recall that all constraints we consider here are of the form $\kappa = \forall\bar{x}(\varphi(\bar{x}) \to \psi(\bar{x}))$, where ψ may contain existential quantifiers. We also often omit universal quantification, simply writing $\kappa = \varphi(\bar{x}) \to \psi(\bar{x})$. This includes TGDs, EGDs, and DCs as well, since $\neg\varphi$ is of course $\varphi \to \bot$. A violation of such a constraint is an instantiation $\bar{a}$ of free variables that makes the implication false, i.e., $\varphi(\bar{a})$ is true but $\psi(\bar{a})$ is false. It will be convenient to formally define violations in terms of homomorphisms. Recall that in all the constraints $\kappa = \varphi(\bar{x}) \to \psi(\bar{x})$, the formula φ is a conjunction of atoms, and thus can be viewed as a set of atoms over constants and variables in $\bar{x}$. An assignment of values $\bar{a}$ to variables $\bar{x}$ making $\varphi(\bar{a})$ true is then a homomorphism $h : dom(\varphi) \to dom(D)$. We let $h(\kappa)$ be κ in which every variable x from $\bar{x}$ is replaced with $h(x)$.

Definition 3.2. (**Constraint Violation**) For a database D, a D-*violation* of a constraint $\kappa = \varphi \to \psi$ is a homomorphism $h : dom(\varphi) \to dom(D)$ such that D does not satisfy $h(\kappa)$. We denote the set of D-violations of κ by $\mathsf{V}(D, \kappa)$ and for a set Σ of constraints we write $\mathsf{V}(D, \Sigma)$ for $\{(\kappa, h) \mid \kappa \in \Sigma \text{ and } h \in \mathsf{V}(D, \kappa)\}$. ∎

Therefore, $(\kappa, h) \in \mathsf{V}(D, \Sigma)$ means that one of the reasons why the database D is inconsistent with Σ is because it violates $\kappa \in \Sigma$ due to the homomorphism h. A (D, Σ)-operation op is (D', Σ)-*fixing*, for $D' \subseteq \mathsf{B}(D, \Sigma)$, if $\mathsf{V}(D', \Sigma) - \mathsf{V}(op(D'), \Sigma) \neq \varnothing$. Sequences of such operations will satisfy the minimal requirement **req1**.

While fixing operations meet the minimal requirement, in the concrete case of TGDs, EGDs and DCs, they may add or remove facts without any justification, as shown below.

Example 3.3. Let $D = \{R(a, b), R(a, c), T(a, b)\}$ and consider the set $\Sigma = \{\sigma, \eta\}$ of constraints

$$\sigma = R(x, y) \to \exists z\, S(x, y, z)$$
$$\eta = R(x, y), R(x, z) \to y = z.$$

Clearly, the operation $op_1 = +\{S(a, b, c), S(a, a, a)\}$ is fixing since it eliminates from $\mathsf{V}(D, \Sigma)$ the violation (σ, h), where $h = \{x \mapsto a, y \mapsto b\}$. However, there is no justification for adding the fact $S(a, a, a)$. Actually, the operation $+S(a, b, c)$, which adds fewer atoms than op_1, is still fixing.

Another example of an operation that is fixing but unjustified is $op_2 = -\{R(a, b), T(a, b)\}$. The above operation is fixing since it removes from $\mathsf{V}(D, \Sigma)$ the pairs (σ, h_1), (η, h_2) and (η, h_3), where $h_1 = \{x \mapsto a, y \mapsto b\}$, $h_2 = \{x \mapsto a, y \mapsto b, z \mapsto c\}$ and $h_3 = \{x \mapsto a, y \mapsto c, z \mapsto b\}$. Nevertheless, there is no justification for removing the fact $T(a, b)$ since it does not contribute in any of

the above D-violations of Σ. An example of a justified operation that resolves (σ, h_1) is $-R(a,b)$. Moreover, there are three justified operations that resolve (η, h_2) and (η, h_3), namely $-R(a,b)$, $-R(a,c)$ and $-\{R(a,b), R(a,c)\}$. ∎

From the above discussion, we conclude that operations should not only be fixing, but also add as few facts as possible in order to fix a violation, and delete a set of facts only if it contributes as a whole in a violation. Such operations are called *justified*.

Definition 3.4. (**Justified Operation**) Let D be a database and Σ a set of constraints. For a database $D' \subseteq \mathsf{B}(D,\Sigma)$, an operation $op \in \{+F, -F\}$ is called (D',Σ)-*justified* if there exists $(\kappa, h) \in \mathsf{V}(D',\Sigma) - \mathsf{V}(op(D'),\Sigma)$ such that for every non-empty set $G \subsetneq F$

(1) if $op = +F$ then $(\kappa, h) \in \mathsf{V}(+G(D'),\Sigma)$
(2) if $op = -F$ then $(\kappa, h) \notin \mathsf{V}(-G(D'),\Sigma)$. ∎

Note that a (D',Σ)-justified operation is (D',Σ)-fixing since $\mathsf{V}(D',\Sigma) - \mathsf{V}(op(D'),\Sigma) \neq \emptyset$. For TGDs, EGDs, and DCs, these operations can be described as follows.

Proposition 3.5. *Let D be a database, Σ a set of constraints, and $D' \subseteq \mathsf{B}(D,\Sigma)$. If op is (D',Σ)-justified, then there exists $(\kappa, h) \in \mathsf{V}(D',\Sigma) - \mathsf{V}(op(D'),\Sigma)$ such that*

- *If $\kappa = \varphi(\bar{x},\bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x},\bar{z})$ and $op = +F$, then $F = h'(\psi(\bar{x},\bar{z})) - D'$, for some extension h' of h.*
- *If κ is an arbitrary constraint of the form $\varphi \rightarrow \psi$ and $op = -F$, then $F \subseteq h(\varphi)$.*

This simply says that justified operations are very cautious with additions: they add the minimal set of atoms necessary to satisfy a TGD. Notice that single-atom insertions may not be enough to satisfy a multi-head TGD in a single step, and thus, it is crucial to allow the addition of sets of atoms. On the other hand, justified operations are less cautious with deletions, in the sense that they do not try to minimize the number of atoms that need to be removed. This reflects the discussion in the introduction that removing a set of facts that collectively contribute to a violation is a justified operation in constructing repairs, due to the fact that we do not know a priori which atoms should be eliminated, and are, therefore, forced to explore all the possible ways.

Repairing Sequences of Operations

The core idea of our operational approach is that one repeatedly applies justified operations starting from an inconsistent database D. Justified operations ensure requirement **req1**, i.e., every single step in isolation is justified. However, due to potential interaction of operations, they may break requirement **req2** that previously eliminated violations cannot be reintroduced. It is easy to construct examples showing how an addition may reintroduce a violation that has been previously eliminated by a deletion, and vice versa.

We could simply impose **req2** on repairing sequences and then consider those of them that lead to a consistent database. However, as discussed in Section 1, these are minimal requirements that assume nothing about the structure of update operations, databases, and constraints. For our concrete data model and operations, these minimal requirements may be supplemented by others. We now consider two of them, motivated by examples below.

Example 3.6. Consider the database D and key constraint η of Example 3.3, and let $\Sigma' = \{\sigma, \eta\}$, where $\sigma = T(x,y) \rightarrow R(x,y)$. Assume that we apply $-\{R(a,b), R(a,c)\}$ followed by $+R(a,b)$. It is easy to check that this sequence satisfies **req1**, **req2**, and leads to a consistent database. Nevertheless, we would like to rule out such a sequence since it has two conflicting operations $-R(a,b)$ and $+R(a,b)$. In fact, this sequence is equivalent to a simpler one consisting just of $-R(a,c)$. We would like to rule out sequences in which an operation *cancels* the effect of another one, i.e., impose the following condition:

- *No Cancellation:* A fact that has been added (removed) should not be removed (added) later. ∎

Our next example shows that in a sequence of justified operations, which lead to a consistent database, a fact that has been added may become unjustified. We should not consider such sequences as repairing ones since a fact should be added only if it is really needed to satisfy a constraint.

Example 3.7. Consider the database D and the set $\Sigma = \{\sigma, \eta\}$ of constraints given in Example 3.3. Assume that we apply the operations $+S(a,b,c)$ and $-R(a,b)$ in this order. This sequence satisfies **req1** and **req2**, and can easily be extended (e.g., with $+S(a,c,b)$) to lead to a consistent database. However, the reason why $S(a,b,c)$ has been added was the existence of $R(a,b)$, or, in other words, the violation (σ, h), where $h = \{x \mapsto a, y \mapsto b\}$. But, after the removal of $R(a,b)$ by the second operation, $S(a,b,c)$ is not needed anymore. Thus, its existence becomes unjustified. We want to forbid this by imposing the following condition:

- *Global Justification of Additions:* The justification for added facts should remain valid. ∎

We proceed to formalize the above properties, and introduce the notion of repairing sequence. Consider a database D and a set Σ of constraints. Given a sequence (perhaps even infinite) $s = (op_i)_{i \geq 1}$ of (D,Σ)-operations, we define

$$D_0^s = D \quad \text{and} \quad D_i^s = op_i(\cdots op_1(D)\cdots) \text{ for } i > 0.$$

In other words, D_i^s is obtained by applying to D the first i operations of s. The notion of repairing sequence follows:

Definition 3.8. (**Repairing Sequence**) Consider a database D and a set Σ of constraints. A sequence of (D,Σ)-operations $s = (op_i)_{i \geq 1}$ is called (D,Σ)-*repairing* if it satisfies **req1** and **req2**, and for every $i \geq 1$:

(1) (*Local Justification*) op_i is (D_{i-1}^s, Σ)-justified.
(2) (*No Cancellation*) $op_i = +F$ and $op_j = -G$ implies $F \cap G = \emptyset$, for every $i \neq j$.
(3) (*Global Justification of Additions*) For every $j > i$, $op_i = +F$ implies op_i is $(D_{i-1}^s - H, \Sigma)$-justified, with $H = \bigcup_{i<k\leq j,\, op_k = -G} G$.

Let $\mathsf{RS}(D,\Sigma)$ be the set of all (D,Σ)-repairing sequences. ∎

We now establish some crucial properties of repairing sequences.

Proposition 3.9. *Consider a database D, a set Σ of constraints, and $s = (op_i)_{i\geq 1} \in \mathsf{RS}(D,\Sigma)$. It holds that s and $\mathsf{RS}(D,\Sigma)$ are finite.*

Finiteness of repairing sequences is an immediate corollary of **req1**, **req2**, and the fact that the number of violations is finite.

Actually, it is implicit in the proof of Proposition 3.9 that the length of a repairing sequence is polynomial in the size of D. Finiteness of repairing sequences, together with the fact that the number of operations is finite, implies finiteness of $\mathrm{RS}(D, \Sigma)$. For a repairing sequence $s = (op_i)_{1 \le i \le n}$, we define its result as $s(D) = D_n^s$.

Note that in the definition of repairing sequences we did not insist that the resulting database is consistent, i.e., that $s(D) \models \Sigma$. There are sequences s such that no extension $s \cdot op$ is a (D, Σ)-repairing sequence; we call these sequences *complete*. Complete sequences describe attempts to construct repairs, and attempts can succeed or fail. They succeed if $s(D) \models \Sigma$, i.e., the sequence is *successful*, otherwise it is *failing*. Consider, e.g., the database $D = \{R(a)\}$ and the set of constraints $\Sigma = \{R(x) \rightarrow T(x),\ T(x) \rightarrow \bot\}$. The (D, Σ)-repairing sequence $s = +T(a)$ is failing since s cannot be extended into a repairing sequence and $s(D) \not\models \Sigma$.

Operational Repairs

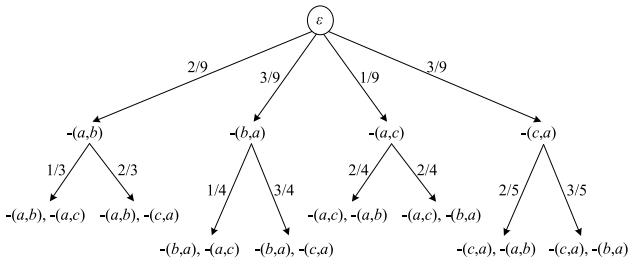
The high-level idea of our approach is to repair an inconsistent database by repeatedly applying justified operations, as long as they give rise to a repairing sequence. This gives us additional flexibility in deciding which updates are more likely than others, while the standard CQA approach declares them all to be equally likely. A formalization of different likelihoods of different operations can be done by using a well known tool from probability theory, namely *Markov chains* [27]. Let us illustrate this via a simple example.

Repairing Sequences in Action. Consider the database

$$D = \{\mathrm{Pref}(a,b), \mathrm{Pref}(a,c), \mathrm{Pref}(a,d), \mathrm{Pref}(b,a), \mathrm{Pref}(b,d), \mathrm{Pref}(c,a)\},$$

and the set Σ that contains a single DC $\mathrm{Pref}(x,y), \mathrm{Pref}(y,x) \rightarrow \bot$, which states that the preference relation over, e.g., products, is not symmetric. It is clear that D is inconsistent w.r.t. Σ. Our goal is to repair D via a (D, Σ)-repairing sequence of operations, which, in this case, are always deletions. However, during the repairing process, we would like to take into account the fact that some products have more support than other ones. For example, a has more support than b since a is preferred more often than b. This is achieved by applying the (D, Σ)-operation $-\mathrm{Pref}(b,a)$ with higher probability than $-\mathrm{Pref}(a,b)$ since it is more likely that a is preferred over b, and thus we would like to keep $\mathrm{Pref}(a,b)$ with higher probability than removing it.

Our intention described above can be nicely captured via a tree-shaped Markov chain like the one shown below. Such a Markov chain is basically a tree encoding all the possible (D, Σ)-repairing sequences that lead to a database that is consistent with Σ:

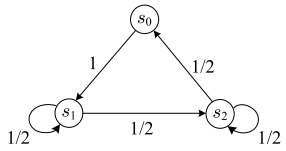


For brevity, we omit the predicate Pref in the above figure, i.e., instead of writing $-\mathrm{Pref}(a,b)$ we simply write $-(a,b)$. The states of M are (D, Σ)-repairing sequences with ε being the empty sequence, which is by definition repairing. The edges are labeled with a probability $p \in [0,1]$, which is simply the probability of moving from one state to another.

Starting from the database D, the probability of removing $\mathrm{Pref}(b,a)$ is $3/9$, and the probability of removing $\mathrm{Pref}(a,b)$ is $2/9$. This captures our intention of keeping $\mathrm{Pref}(a,b)$ with higher probability than removing it since a has more support than b in D. In fact these probabilities are not arbitrary but rather are provided by a precise algorithm that extracts them from the data; this will be explained in Example 3.11. Analogously, since a has more support than c in D, the probability of removing $\mathrm{Pref}(c,a)$ is higher than the probability of removing $\mathrm{Pref}(a,c)$. Now, assume that we choose to apply $-\mathrm{Pref}(b,a)$, and thus construct the database $D' = D - \{\mathrm{Pref}(b,a)\}$. The probability of removing $\mathrm{Pref}(c,a)$ is $3/4$, while the probability of removing $\mathrm{Pref}(a,c)$ is $1/4$, which again captures our intention of keeping $\mathrm{Pref}(a,c)$ with higher probability than removing it since a has more support than c in D'. Observe that the leaves of M are repairing sequences s such that $s(D) \models \Sigma$. These sequences are complete, and thus cannot be extended further. Since in a Markov chain the probabilities of the outgoing edges of a state must sum up to one, every leaf of M has an implicit outgoing edge connecting it to itself with probability 1. Having the Markov chain M in place, we can assign probabilities to repairs. For example, the probability of the repair $D - \{\mathrm{Pref}(b,a), \mathrm{Pref}(c,a)\}$ is $\frac{3}{9} \cdot \frac{3}{4} + \frac{3}{9} \cdot \frac{3}{5} = 0.45$, which is the probability that the initial state ε reaches the state $-(b,a), -(c,a)$ plus the probability that ε reaches the state $-(c,a), -(b,a)$.

As we shall see in the next section, having repairs with probabilities allows us to talk about the probability with which a consistent answer is entailed. We now proceed with the formalizations of our approach. But first we need to recall the basics on Markov chains.

The Basics on Markov Chains. A Markov chain is essentially an edge-labeled directed graph, where the nodes are its states and the edges are labeled with a probability $p \in [0,1]$ so that for each node s, the labels of its outgoing edges sum up to 1. An edge (s, s') with label p says that with probability p, the state changes from s to s'. The goal is to answer questions of the form: what is the probability that, starting from a certain state s, we reach state s' after k steps? Consider, for example, the Markov chain over the state space $\{s_0, s_1, s_2\}$ shown below:



It is naturally encoded as a matrix

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \end{bmatrix}$$

where in each row entries add up to 1. If we want to know the probability of reaching s_2 from s_1 after 3 steps, this is simply $\mathbf{P}^3(s_1, s_2) = 3/8$, i.e., the $(1,2)$-th cell of the matrix obtained by

multiplying $\mathbf{P}$ three times (that is, the sum of the probabilities of all paths of length 3 from s_1 to s_2).

Formally, a *Markov chain* M over a (finite) state space $S = \{s_0, \ldots, s_k\}$ is a pair $(s_0, \mathbf{P})$, where s_0 is the initial state of M and $\mathbf{P} : S \times S \to [0, 1]$ is a stochastic function, i.e., $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ for every state $s \in S$. Since S is finite, the function $\mathbf{P}$ can be naturally seen as an $|S| \times |S|$ matrix, called *probability transition matrix*, whose (i, j)-th cell contains $\mathbf{P}(s_i, s_j)$. By abuse of notation, whenever $\mathbf{P}$ is treated as a matrix, we write $\mathbf{P}(s_i, s_j)$ instead of $\mathbf{P}(i, j)$. Starting from the state s_i, the probability of reaching s_j after n steps is $\mathbf{P}^n(s_i, s_j)$, where $\mathbf{P}^n = \prod_{i=1}^{n} \mathbf{P}$.

A state $s \in S$ is called *absorbing* if $\mathbf{P}(s, s) = 1$, i.e., s is reachable from itself with probability 1. The set of *reachable absorbing states* of M is the set of absorbing states of M that are reachable from s_0 with non-zero probability, i.e., the set

$$\mathsf{ras}(M) = \left\{ s \in S \;\middle|\; \begin{array}{c} \mathbf{P}(s, s) = 1 \text{ and} \\ \exists n \text{ such that } \mathbf{P}^n(s_0, s) > 0 \end{array} \right\}.$$

Finally, a key notion is the *hitting distribution* of M, defined as the limit $\lim_{n \to \infty} \mathbf{P}^n(s_0)$ if it exists, where $\mathbf{P}^n(s_0)$ is the 0-th row of $\mathbf{P}^n$; otherwise, we say that M does not admit a hitting distribution. Intuitively, the hitting distribution describes the long-term behavior of the Markov chain.

Repairs via Markov Chains. We now formalize the idea of assigning likelihoods to operations extending sequences: for all possible extensions $s \cdot op_1, \ldots, s \cdot op_k$ of a repairing sequence s, we assign probabilities $p_1, \ldots, p_k$ to them so they add up to 1. This is done by exploiting a tree-shaped Markov chain that arranges its states (i.e., repairing sequences) in a tree, where the children of each state are its possible extensions. Furthermore, states corresponding to complete sequences, i.e., cannot be extended, coincide with the absorbing states of the Markov chain. Formally, let ε be the empty sequence of operations, which is by definition repairing.

Definition 3.10. (**Repairing Markov Chain**) For a database D and a set Σ of constraints, a (D, Σ)-*repairing Markov chain* is a Markov chain of the form $(\varepsilon, \mathbf{P})$, where $\mathbf{P} : \mathsf{RS}(D, \Sigma) \times \mathsf{RS}(D, \Sigma) \to [0, 1]$ is such that:

(1) For each sequence $s \in \mathsf{RS}(D, \Sigma)$, s is complete iff it is absorbing, i.e., $\mathbf{P}(s, s) = 1$.
(2) If $s, s' \in \mathsf{RS}(D, \Sigma)$ are distinct, then $\mathbf{P}(s, s') > 0$ implies $s' = s \cdot op$ for some (D, Σ)-operation op.

A *repairing Markov chain generator* w.r.t. Σ is a function M_Σ assigning to every database D a (D, Σ)-repairing Markov chain. ∎

The purpose of the repairing Markov chain generator is to provide a generic mechanism for defining a family of repairing Markov chains independently of the input database. This means that one can design a repairing Markov chain generator M_Σ once, and whenever the database D changes, the desired repairing Markov chain is obtained by simply applying M_Σ on D. Of course, the way the repairing Markov chain generator is designed depends on the specific application. For instance, in our preference example given above, our intention is to assign probabilities to operations in such a way that the repairing process reflects the fact that some products have more support than other ones in the input database. Instead of devising a (D, Σ)-repairing Markov chain that captures our intention on the particular database D, as we did above, one can design a repairing Markov chain generator M_Σ such that, for every input database D over the schema $\{\text{Pref}\}$, $M_\Sigma(D)$ is the desired repairing Markov chain that captures our intention on D.

Example 3.11. Consider again the product preferences scenario, and let Σ be the set that contains the single denial constraint which states that the preference relation is not symmetric. We define a repairing Markov chain generator M_Σ such that, for every database D over $\{\text{Pref}\}$, $M_\Sigma(D)$ is the repairing Markov chain that we are looking for. Let the weight $w(\alpha, D)$ of an atom $\alpha = \{\text{Pref}(a, b)\}$ in a database D be the number of facts $\text{Pref}(a, \cdot)$ in D, i.e., where a is preferred. Assuming that $V_\Sigma(D)$ collects all the atoms involved in a violation of Σ by D, the *importance* $I_\Sigma(\alpha, D)$ of an atom $\alpha \in D$ is defined as the relative weight of α w.r.t. all the atoms involved in a violation, i.e., $I_\Sigma(\alpha, D) = w(\alpha, D) / \sum_{\beta \in V_\Sigma(D)} w(\beta, D)$. We define the probability of removing an atom $\alpha = \text{Pref}(a, b)$ as the importance of its symmetric atom $\bar{\alpha} = \text{Pref}(b, a)$. Formally, the repairing Markov chain $M_\Sigma(D) = (\varepsilon, \mathbf{P})$ is such that for $s, s' \in \mathsf{RS}(D, \Sigma)$,

$$\mathbf{P}(s, s') = \begin{cases} 1 & \text{if } s = s' \text{ and } s \text{ is complete} \\ I_\Sigma(\bar{\alpha}, s(D)) & \text{if } s' = s \cdot -\alpha, \text{ for } \alpha \in D \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to verify that this gives us a Markov chain (probabilities sum up to 1), and if we consider the database D used in our preference example above, then $M_\Sigma(D)$ is precisely the (D, Σ)-repairing Markov chain depicted in the figure. ∎

The aim of the above simple was to illustrate the key notion of the repairing Markov chain generator. To demonstrate the existence of appropriate Markov chain generators in realistic scenarios, we give another, slightly more involved example, based on the standard scenario of data integration. In such a scenario, different (possibly conflicting) facts, which are coming from different sources, have to be integrated into a single unified database. In this setting, it is common to assign to each fact a level of trust that depends on the source it is coming from. Such a level of trust can be computed by applying one of the many methodologies that have been proposed in the literature; see, e.g., [16, 41].

Example 3.12. Consider the schema $\mathbf{S} = \{R/2\}$ and let Σ contain a single key $R(x, y), R(x, z) \to y = z$ (the discussion easily generalizes to an arbitrary set of keys). Let D be a database over $\mathbf{S}$ that has been obtained by integrating data from different sources, where each fact $\alpha \in D$ is assigned a level of trust denoted by $tr(\alpha) \in [0, 1]$. Given two facts $\alpha, \beta \in D$, we define the *relative trust* of α w.r.t. β as $tr_{\alpha|\beta} = tr(\alpha)/(tr(\alpha) + tr(\beta))$.

Let $V_\Sigma(D) = \{\{\alpha, \beta\} \subseteq D \mid \{\alpha, \beta\} \not\models \Sigma\}$ be the set of all pairs of atoms violating Σ. There are three possible ways of fixing the violations due to $\{\alpha, \beta\} \in V_\Sigma(D)$: either remove only α, or remove only β, or remove both. Our intention is as follows: $tr_{\alpha|\beta} < tr_{\beta|\alpha}$ implies that the operation $-\alpha$ should have a higher probability than $-\beta$. Furthermore, the probability of $-\{\alpha, \beta\}$ should be lower than the probability of $-\alpha$ and the probability of $-\beta$. To formally capture our intention via a repairing Markov chain generator, we first define the weight of each operation trying to fix the violations

due to $\{\alpha, \beta\} \in V_\Sigma(D)$. The weight $w_{\alpha,\beta}(-F)$ of applying $-F$ is equal to 0 if $F \not\subseteq \{\alpha, \beta\}$, otherwise

$$\begin{aligned} w_{\alpha,\beta}(-\alpha) &= tr_{\beta|\alpha} \cdot (1 - tr_{\alpha|\beta} \cdot tr_{\beta|\alpha}) \\ w_{\alpha,\beta}(-\beta) &= tr_{\alpha|\beta} \cdot (1 - tr_{\alpha|\beta} \cdot tr_{\beta|\alpha}) \\ w_{\alpha,\beta}(-\{\alpha,\beta\}) &= (1 - tr_{\alpha|\beta}) \cdot (1 - tr_{\beta|\alpha}). \end{aligned}$$

The first (resp., second) formula encodes the event of trusting β (resp., α) but not both α and β, while the last formula encodes the event of trusting neither α nor β. Notice that for every $\{\alpha, \beta\} \in V_\Sigma(D)$, the three weights above sum up to one, i.e., they encode relative weights of each operation fixing a violation due to $\{\alpha, \beta\}$.

To devise our repairing Markov chain generator M_Σ, we normalize the weights of each set $\{\alpha, \beta\} \in V_\Sigma(D)$ w.r.t. all the sets in $V_\Sigma(D)$. For every database D over the schema $\mathbf{S}$, $M_\Sigma(D) = (\varepsilon, \mathbf{P})$ is such that for $s, s' \in \mathsf{RS}(D, \Sigma)$,

$$\mathbf{P}(s, s') = \begin{cases} 1 & \text{if } s = s' \text{ is complete} \\ \dfrac{\sum_{\{\alpha,\beta\} \in V_\Sigma(s(D))} w_{\alpha,\beta}(-F)}{|V_\Sigma(s(D))|} & \text{if } s' = s \cdot -F \\ 0 & \text{otherwise.} \end{cases}$$

This completes the definition of M_Σ. ∎

We proceed to define our notion of operational repair: they are obtained by repairing sequences that are reachable absorbing states of a repairing Markov chain.

Definition 3.13. (**Operational Repair**) Given a database D, a set Σ of constraints, and a repairing Markov chain generator M_Σ w.r.t. Σ, an *(operational) repair* of D w.r.t. M_Σ is a consistent database of the form $s(D)$, where s is a reachable absorbing state of $M_\Sigma(D)$, i.e., $s \in \mathsf{ras}(M_\Sigma(D))$. ∎

If we consider again the database D and the set Σ in our preference example above, it is easy to see that D has four repairs w.r.t. M_Σ, where M_Σ is the Markov chain generator defined in Example 3.11. In particular, we get the following repairs:

$D - \{\mathsf{Pref}(a, b), \mathsf{Pref}(a, c)\}$ with probability $2/9 \cdot 1/3 + 1/9 \cdot 2/4$
$D - \{\mathsf{Pref}(a, b), \mathsf{Pref}(c, a)\}$ with probability $2/9 \cdot 2/3 + 3/9 \cdot 2/5$
$D - \{\mathsf{Pref}(b, a), \mathsf{Pref}(a, c)\}$ with probability $3/9 \cdot 1/4 + 1/9 \cdot 2/4$
$D - \{\mathsf{Pref}(b, a), \mathsf{Pref}(c, a)\}$ with probability $3/9 \cdot 3/4 + 3/9 \cdot 3/5$

An operational repair may be obtainable via multiple repairing sequences that are reachable absorbing states of the underlying repairing Markov chain. To calculate the probability of a repair D', we have to sum up the probabilities of all reachable absorbing states s so that $D' = s(D)$. These must come from the hitting distribution, but in general Markov chains may not admit it. Fortunately, in our case the hitting distribution always exists since a repairing Markov chain is a finite tree-like structure, and thus, there is always a finite path leading to an absorbing state.

Proposition 3.14. *Given a database D and a set Σ of constraints, every (D, Σ)-repairing Markov chain $(\varepsilon, \mathbf{P})$ admits a hitting distribution, i.e., $\lim_{n\to\infty} \mathbf{P}^n(\varepsilon)$ exists.*

The existence of a hitting distribution allows us to define the probability of an operational repair D' of D w.r.t. M_Σ as

$$\mathsf{P}_{D,M_\Sigma}(D') = \sum_{s \in \mathsf{ras}(M_\Sigma(D)) \text{ and } D' = s(D)} \pi(s),$$

where π is the hitting distribution of $M_\Sigma(D)$. For a database $D' \subseteq \mathsf{B}(D, \Sigma)$ that is not an operational repair, we let $\mathsf{P}_{D,M_\Sigma}(D') = 0$. Finally, we define our *semantics of an inconsistent database* as the set of repair-probability pairs

$$[\![D]\!]_{M_\Sigma} = \left\{ (D', \mathsf{P}_{D,M_\Sigma}(D')) \;\middle|\; \begin{array}{c} D' \subseteq \mathsf{B}(D, \Sigma), \\ \mathsf{P}_{D,M_\Sigma}(D') > 0 \end{array} \right\}.$$

We conclude this section by comparing operational repairs with the standard repairs of Arenas, Bertossi, and Chomicki [2] (see the definition of the ABC semantics $[\![D]\!]^{\mathsf{ABC}}_\Sigma$ in Section 2). Of course operational repairs depend on the underlying Markov chain; it may be set up in such a way that some of the ABC repairs are not reached (for example, by assigning probability 0 to deletions, we will not be able to repair key violations). However, we can always find a Markov chain so that operational repairs with respect to it include all ABC repairs. Let M^u_Σ be the uniform Markov chain generator: if for $s \in \mathsf{RS}(D, \Sigma)$ its possible extensions in $\mathsf{RS}(D, \Sigma)$ are exactly $s_1 = s \cdot op_1, \ldots, s_k = s \cdot op_k$, then $\mathbf{P}(s, s_i) = 1/k$ for all $i \leq k$.

Proposition 3.15. *Let D be a database and Σ a set of constraints, and let $D' \in [\![D]\!]^{\mathsf{ABC}}_\Sigma$. Then D' is an operational repair of D w.r.t. M^u_Σ.*

4 OPERATIONAL CONSISTENT QUERY ANSWERING

We are targeting a more refined approach to query answering compared to the usual certain answers: its goal is to compute the probability that a tuple is in the answer. For a database D, a set of constraints Σ, a Markov chain generator M_Σ, a query $Q(\bar{x})$ and a tuple $\bar{t}$ of constants of arity $|\bar{x}|$, we define the conditional probability of $\bar{t}$ being in the answer to Q over some operational repair as:

$$\mathsf{CP}_{D,M_\Sigma,Q}(\bar{t}) = \frac{\sum_{(D',p) \in [\![D]\!]_{M_\Sigma} \text{ and } \bar{t} \in Q(D')} p}{\sum_{(D',p) \in [\![D]\!]_{M_\Sigma}} p}$$

if at least one operational repair exists (and thus the denominator is not zero); otherwise, $\mathsf{CP}_{D,M_\Sigma,Q}(\bar{t}) = 0$. For brevity, if D, M_Σ, and Q are clear from the context, we simply write $\mathsf{CP}(\bar{t})$. Note that failing repairing sequences will be assigned a non-zero probability in the hitting distribution, but they should not contribute towards the probability that a tuple is in the answer. The conditional probability accounts for this by normalizing the probability of the tuple being in the answer over the probabilities of successful repairing sequences. The definition of consistent answers follows.

Definition 4.1. (**Operational Consistent Answers**) For a database D, a set Σ of constraints, a repairing Markov chain generator M_Σ, and a query $Q(\bar{x})$, the set of *operational consistent answers* to Q w.r.t. D and M_Σ is defined as the set of tuple-probability pairs

$$\mathsf{OCA}_{M_\Sigma}(D, Q) = \left\{ (\bar{t}, \mathsf{CP}_{D,M_\Sigma,Q}(\bar{t})) \mid \bar{t} \in dom(\mathsf{B}(D, \Sigma))^{|\bar{x}|} \right\}.$$ ∎

Example 4.2. We use the database D from our preference example and the repairing Markov chain generator M_Σ from Example 3.11.

Consider a query $Q(x)$ stating that x is the most preferred product, i.e., $\forall y\ (\text{Pref}(x, y) \vee x = y)$. The set of the certain answers to Q under the ABC semantics is empty. Indeed, in three out of four repairs (see above) such most preferred product does not exist. In the last of the repairs, $\{\text{Pref}(a, b), \text{Pref}(a, c), \text{Pref}(a, d), \text{Pref}(b, d)\}$, such a product does exist, namely a. Thus, $\text{OCA}_{M_\Sigma}(D, Q) = \{(a, 0.45)\}$. This information on the degree of certainty that a is preferred over all the other products is something that the traditional CQA approach cannot provide us with. ∎

Complexity Analysis

We now study the complexity of computing operational consistent answers. We focus on the following function problem, called *operational consistent query answering:*

PROBLEM :	OCQA
INPUT :	A database D, a set of constraints Σ, a repairing MC generator M_Σ w.r.t. Σ, a query $Q(\bar{x})$, and $\bar{t} \in dom(\text{B}(D, \Sigma))^{\lvert\bar{x}\rvert}$.
OUTPUT :	$\text{CP}_{D, M_\Sigma, Q}(\bar{t})$.

Let us clarify that this general formulation refers to the combined complexity of the problem. We are interested in the *data complexity*, i.e., complexity of the problem $\text{OCQA}(\Sigma, M_\Sigma, Q)$ when Σ, M_Σ, and Q are fixed, and only the database D and the tuple $\bar{t}$ form the input. We adopt the convention that when we talk about the data complexity of a problem like OCQA (i.e., the class of problems $\text{OCQA}(\Sigma, M_\Sigma, Q)$), we say that it is complete for a class $\mathcal{K}$ if each of the problems $\text{OCQA}(\Sigma, M_\Sigma, Q)$ is in $\mathcal{K}$, and there is one problem $\text{OCQA}(\Sigma, M_\Sigma, Q)$ that is $\mathcal{K}$-hard.

Before giving the complexity of OCQA, we need to clarify a technical issue related to the presentation of Markov chain (generators). We assume that they are *well-behaved*. For a (D, Σ)-repairing Markov chain $(\varepsilon, \mathbf{P})$ this means that $\mathbf{P}$ is a function computable in polynomial time w.r.t. the size of D. Note that this implies that the problem of checking whether a (D, Σ)-repairing sequence is complete is solvable in polynomial time w.r.t. D (see condition 1 in Definition 3.10). We can formally show that checking whether a (D, Σ)-repairing sequence is complete is indeed solvable in polynomial time in D, which means that assuming $\mathbf{P}$ is computable in polynomial time is not unrealistic.

A Markov chain generator M_Σ w.r.t. Σ is well-behaved if for every database D, $M_\Sigma(D)$ is computable in polynomial time w.r.t. the size of D, and there is a polynomial $f(\cdot)$ such that $M_\Sigma(D) = (\varepsilon, \mathbf{P})$ is well-behaved and the probabilities computed by $\mathbf{P}$ have a common denominator encoded using $f(|D|)$ bits, i.e., there exists a natural number $d > 0$ of $f(|D|)$ bits such that each probability computed by $\mathbf{P}$ is of the form k/d, where $0 \leq k \leq d$ and $k \in \mathbb{N}$. In the absence of the well-behavedness assumption, the complexity of OCQA of course may increase, but this will be due to reasons that have nothing to do with consistent query answering but rather with the internal representation of Markov chains.

Returning to the OCQA problem, it can be solved in polynomial time having access to a #P oracle. Recall that #P is the class of function problems that ask for the number of solutions to an NP problem. Recall also that FP is the class of function problems that can be solved in polynomial time.

Theorem 4.3. OCQA *is* $\text{FP}^{\#P}$*-complete in data complexity. The hardness holds even for inclusion dependencies or keys or denial constraints, and conjunctive queries.*

The hardness is shown by a reduction from query answering over probabilistic databases [14]. Notice that in [14] it is shown that the latter problem is #P-hard. However, it is known that a problem is #P-hard iff it is $\text{FP}^{\#\text{P}}$-hard; see, e.g., [1].

Let us now discuss the upper bound. It is implicit in [37] that $\text{FP}^{\#\text{P}} = \text{FP}^{\#\Sigma_3^P}$, where $\#\Sigma_3^P$ is the class of function problems that ask for the number of solutions to a Σ_3^P problem. Therefore, it suffices to show that OCQA is in $\text{FP}^{\#\Sigma_3^P}$ in data complexity. Consider an instance $(D, \Sigma, M_\Sigma, Q, \bar{t})$ of OCQA. By employing a padding argument, we first show that there exist two decision problems Π_1 and Π_2, which accept as input $D, \Sigma, M_\Sigma, Q, \bar{t}$ such that

$$\frac{\#sol(\Pi_1)}{d^n} = \sum_{(D', p) \in [\![D]\!]_{M_\Sigma} \text{ and } \bar{t} \in Q(D')} p$$

and

$$\frac{\#sol(\Pi_2)}{d^n} = \sum_{(D', p) \in [\![D]\!]_{M_\Sigma}} p,$$

where $\#sol(\Pi)$ is the number of solutions to the problem Π, d is the common denominator of the probabilities computed by $M_\Sigma(D)$, which exists since M_Σ is well-behaved, and n is the maximum length of a (D, Σ)-repairing sequence, which, by Proposition 3.9, is finite.

We then show that Π_1 and Π_2 are in Σ_3^P if Σ, M_Σ, Q are fixed. This relies on the fact that $M_\Sigma(D) = (\varepsilon, \mathbf{P})$ can be computed in polynomial time, $\mathbf{P}$ is a polynomial-time computable function, and d can be encoded using polynomially many bits. We also exploit the fact that checking whether a sequence s belongs to $\text{RS}(D, \Sigma)$ is solvable in polynomial time. Finally, we use the fact that n is polynomial in the size of D and $|\text{RS}(D, \Sigma)|$ is exponential in D, which is implicit in the proof of Proposition 3.9.

Having Π_1 and Π_2 in place, we can easily solve our problem in polynomial time by applying the following algorithm:

(1) Compute the numbers $\#sol(\Pi_1)$ and $\#sol(\Pi_2)$.
(2) If $\#sol(\Pi_2) > 0$, then return $\frac{\#sol(\Pi_1)}{\#sol(\Pi_2)}$; else, return 0.

Step 1 can be done in $O(1)$ time via a $\#\Sigma_3^P$ oracle. Note that $\#sol(\Pi_1)$ and $\#sol(\Pi_2)$ are, in general, exponential numbers, and can be represented using polynomially many bits. Thus, $\frac{\#sol(\Pi_1)}{\#sol(\Pi_2)}$ can be computed in polynomial time, and Theorem 4.3 follows.

Thus, OCQA cannot be efficiently solved with respect to data complexity. This of course is not surprising since operational repairs can encode the standard ones of [2], and for them the problem of finding certain answers (i.e., insisting on probability 1) can be coNP-hard. The main contribution of Theorem 4.3 is pinpointing the exact complexity of the problem. The key advantage of our flexible approach to consistent query answering is that it allows us to talk about *approximations* as opposed to exact query answering. This is the subject of the next section.

5 TOWARDS EFFICIENT APPROXIMATIONS

When a problem is computationally hard, it is natural to approximate it. Our problem is $\text{FP}^{\#\text{P}}$-complete, and its output is a number, namely $\text{CP}_{D,M_\Sigma,Q}(\bar{t})$. In general, approximations to such problems can be computed by randomized algorithms A with some probabilistic guarantees. Such algorithms, in addition to their input, receive a stream of random bits that they can use for the purpose of making random choices. The output of a randomized algorithm A, with input x, is a random variable $A(x)$ that, in our case, should be close to $\text{CP}(\bar{t})$ with high probability. The guarantees come in two shapes: *multiplicative* and *additive* errors [38].

In our case, assume that algorithm A takes D and $\bar{t}$ as an input, with a set of constraints Σ, a Markov chain generator M_Σ, and a query Q fixed. It also takes as input two numbers $\epsilon > 0$ and $0 < \delta < 1$. Multiplicative error means that

$$\Pr\big(|A(D,\bar{t},\epsilon,\delta) - \text{CP}(\bar{t})| \leq \epsilon \cdot \text{CP}(\bar{t})\big) \quad \geq \quad 1-\delta$$

where Pr is the probability of an event, while additive error means

$$\Pr\big(|A(D,\bar{t},\epsilon,\delta) - \text{CP}(\bar{t})| \leq \epsilon\big) \quad \geq \quad 1-\delta.$$

The algorithm is required to run in time polynomial in D, $\bar{t}$, $1/\epsilon$, and $\log(1/\delta)$. The parameters ϵ and δ tell us how certain we are that the randomized algorithm provides a good approximation.

In the case of multiplicative error such an algorithm is known as a fully polynomial-time randomized approximation scheme, or FPRAS. In general, an FPRAS is preferable than an additive error algorithm. The reason is because the *relative* error between the output of an FPRAS and the value we want to approximate is bounded by ϵ. This is not true for additive error algorithms, where only the *absolute* error is bounded by ϵ, while the relative error increases as the value we want to approximate decreases. Nevertheless, additive error algorithms are very useful for our purposes since we are approximating probabilities (i.e., the probability of a tuple being in the query answer). Thus, having a high relative error for tuples with small probability is a reasonable price to pay, since such tuples are much less important than tuples with high probability.

The above discussion suggests to study both multiplicative and additive error algorithms. Our main result is that it is hard to obtain an FPRAS for our problem, but approximation with additive error can be obtained in polynomial time for a large class of Markov chain generators that include (but are not limited to) all constraints we considered here, all first-order queries, and deletion updates.

Probability Spaces and Random Variables. We need to recall a few basic definitions. A *probability space* is a pair $\text{PS} = (\Omega, \pi)$, where Ω is a finite set, called *sample space*, and $\pi : \Omega \to [0,1]$ is a function such that $\sum_{\omega\in\Omega} \pi(\omega) = 1$. A subset $E \subseteq \Omega$ is called an *event*. The probability of an event E, denoted $\Pr(E)$ is defined as $\sum_{\omega\in E} \pi(\omega)$. A *random variable* over PS is a function $X : \Omega \to \mathbb{Q}$. The *probability distribution* of X is a function π_X from the image of X to $[0,1]$ such that $\pi_X(x) = \Pr(X = x)$, where $X = x$ denotes the event $\{\omega \in \Omega \mid X(\omega) = x\}$.

Approximation Via FPRAS

We proceed to show that the problem OCQA does not admit an FPRAS, even for very restricted settings such as keys and conjunctive queries. We fix a set Σ of constraints, a repairing Markov chain generator M_Σ w.r.t. Σ, and a query $Q(\bar{x})$. An FPRAS for $\text{OCQA}(\Sigma, M_\Sigma, Q)$ is a randomized algorithm A that takes an instance $I = (D, \bar{t})$ as well as ϵ and δ, and produces a random variable $A(I,\epsilon,\delta)$ over some sample space Ω[1]. In fact, $A(I,\epsilon,\delta)(\omega)$ is a rational number for $\omega \in \Omega$. Consider the event $E \subseteq \Omega$ given by

$$E \;=\; \{\omega \in \Omega \;\mid\; |A(I,\epsilon,\delta)(\omega) - \text{CP}(\bar{t})| \;\leq\; \epsilon \cdot \text{CP}(\bar{t})\}.$$

Then A is an FPRAS for $\text{OCQA}(\Sigma, M_\Sigma, Q)$ if

$$\Pr(E) \geq 1-\delta$$

and A runs in polynomial time in I, $1/\epsilon$ and $\log(1/\delta)$.

We proceed to show that the OCQA problem does not admit an FPRAS, under the widely accepted complexity assumption that RP $\neq$ NP. Recall that RP is the complexity class of problems that are efficiently solvable by a randomized algorithm with a bounded one-sided error (i.e., the answer may mistakenly be "no") [4].

Theorem 5.1. *Assume RP$\neq$ NP. Then there is a set of constraints Σ, a Markov chain generator M_Σ w.r.t. Σ, and a first-order query Q such that there is no FPRAS for* $\text{OCQA}(\Sigma, M_\Sigma, Q)$. *We can further assume that Σ contains only inclusion dependencies or only key dependencies or only denial constraints, and that Q is a conjunctive query.*

A standard approach for establishing the non-existence of an FPRAS is to show that the decision version of the function problem in question is NP-hard [25]. This is how we prove Theorem 5.1. The decision version of $\text{OCQA}(\Sigma, M_\Sigma, Q)$, which we call *tuple probability checking*, is as follows:

PROBLEM :	$\text{TPC}(\Sigma, M_\Sigma, Q(\bar{x}))$
INPUT :	A database D and $\bar{t} \in dom(\text{B}(D,\Sigma))^{\lvert\bar{x}\rvert}$.
OUTPUT :	Is $\text{CP}_{D,M_\Sigma,Q}(\bar{t}) > 0$?

We show that:

Proposition 5.2. *There exists a set of constraints Σ, a Markov chain generator M_Σ w.r.t. Σ, and a first-order query Q such that* $\text{TPC}(\Sigma, M_\Sigma, Q)$ *is NP-hard. The hardness holds even under the restrictions of Theorem 5.1.*

Assume now that $\text{OCQA}(\Sigma, M_\Sigma, Q)$ admits an FPRAS. We can show that $\text{TPC}(\Sigma, M_\Sigma, Q)$ is in BPP, i.e., the class of decision problems that are efficiently solvable via a randomized algorithm with a bounded two-sided error [4]. Since $\text{TPC}(\Sigma, M_\Sigma, Q)$ is NP-hard, by Proposition 5.2, we conclude that NP $\subseteq$ BPP. But it is well-known that this implies RP = NP [25], and Theorem 5.1 follows.

Approximation With Additive Error

Even though OCQA is not approximable with multiplicative error, we now show that it admits an approximation with additive error. Since the output of OCQA is a probability, this is a strong positive result, as explained above. To obtain it, we need to impose a restriction on repairing Markov chain generators. Recall that $\text{ras}(M)$ is the set of absorbing states of a Markov chain M that are reachable from the initial state with non-zero probability.

[1] There is no restriction on the sample space Ω. As we shall see below, a natural sample space is the set of reachable absorbing states of our repairing Markov chain.

Definition 5.3. A repairing Markov chain generator M_Σ is *non-failing* if, for every database D, the set $\mathsf{ras}(M_\Sigma(D))$ does not contain any failing repairing sequences. ∎

This covers several important cases, in particular, the case where only deletions are used, which has been thoroughly studied in the classical CQA setting [12]. A (D,Σ)-repairing Markov chain $M = (\epsilon, \mathbf{P})$ *supports only deletions* if, for every $s, s' \in \mathsf{RS}(D,\Sigma)$ such that $s \neq s'$, $\mathbf{P}(s,s') > 0$ implies $s' = s \cdot -F$ for some $F \subseteq \mathsf{B}(D,\Sigma)$, i.e., only deletions are assigned non-zero probability. A repairing Markov chain generator M_Σ supports only deletions if $M_\Sigma(D)$ supports only deletions for every database D.

Proposition 5.4. *Consider a set Σ of TGDs, EGDs, and DCs, and a repairing Markov chain generator M_Σ. If M_Σ supports only deletions, then it is non-failing.*

We now show that for non-failing Markov chain generators, OCQA is efficiently approximable up to a polynomial additive error factor. Intuitively, the reason why we focus on such Markov chain generators is that in the absence of a failing repairing sequence, the probability $\mathsf{CP}(\bar{t})$ of a tuple $\bar{t}$ is not a conditional probability anymore since the event of reaching a failing repairing sequence has probability 0. Thus, we have to approximate only the numerator of $\mathsf{CP}(\bar{t})$. The algorithm that we provide does this, but it is not yet clear how it could be used to approximate the ratio that defines the conditional probability where the denominator is less than one.

In the rest of the section, fix a set Σ of constraints, a non-failing repairing Markov chain generator M_Σ w.r.t. Σ, and a first-order query $Q(\bar{x})$. As for the case of multiplicative error, a polynomial-time randomized approximation with additive error for $\mathsf{OCQA}(\Sigma, M_\Sigma, Q)$ is a randomized algorithm A such that, for every instance $I = (D, \bar{t})$ of it, where D is a database and $\bar{t} \in dom(\mathsf{B}(D,\Sigma))^{|\bar{x}|}$, $\epsilon > 0$, and $0 < \delta < 1$, guarantees

$$\Pr(|A(I,\epsilon,\delta) - \mathsf{CP}(\bar{t})| \leq \epsilon) \;\; \geq \;\; 1-\delta,$$

and runs in polynomial time in I, $1/\epsilon$ and $\log(1/\delta)$. In other words, A returns a random variable over some sample space Ω, and the probability of the event

$$\{\omega \in \Omega \;\; | \;\; |A(I,\epsilon,\delta)(\omega) - \mathsf{CP}(\bar{t})| \;\leq\; \epsilon\}$$

is at least $1-\delta$. Notice that the crucial difference compared to the definition of FPRAS for $\mathsf{OCQA}(\Sigma, M_\Sigma, Q)$ is that ϵ is not multiplied by $\mathsf{CP}(\bar{t})$. Our main positive result follows:

Theorem 5.5. *For every set of constraints Σ, non-failing repairing Markov chain generator M_Σ w.r.t. Σ, and first-order query Q, $\mathsf{OCQA}(\Sigma, M_\Sigma, Q)$ admits a polynomial-time randomized approximation with additive error.*

We proceed to establish the above result. The sample space Ω that we use in the proof is the set $\mathsf{ras}(M_\Sigma(D))$ of reachable absorbing states of $M_\Sigma(D)$. Since M_Σ is non-failing, the elements of Ω are successful repairing sequences. The distribution π on Ω is the hitting distribution of $M_\Sigma(D)$.

Assume we have access to a polynomial-time randomized algorithm, which we call Sample, that works as follows. It takes as input an instance $I = (D, \bar{t})$ of $\mathsf{OCQA}(\Sigma, M_\Sigma, Q)$ and outputs a random variable X_I over (Ω, π) that maps Ω to $\{0,1\}$, and $\Pr(X_I = 1) = \mathsf{CP}(\bar{t})$. Under this assumption, we devise a polynomial-time randomized approximation with additive error for $\mathsf{OCQA}(\Sigma, M_\Sigma, Q)$. Once this is done, we provide a formal definition of Sample.

Consider $n > 0$ random variables $X_I^1, \ldots, X_I^n$ that are computed using Sample. Let $X_{I,n}$ be the random variable $1/n \cdot \sum_{i=1}^n X_I^i$, which is the sample mean of $X_I^1, \ldots, X_I^n$. Note that $X_I^1, \ldots, X_I^n$ are defined over the same probability space, they have the same image $\{0,1\}$, and $\Pr(X_I^i = 1) = \mathsf{CP}(\bar{t})$ for every $1 \leq i \leq n$. By Hoeffding's inequality, which provides a lower bound on the probability that the sample mean of n random variables does not deviate from its expected value by more than some factor [24], we obtain that

$$\Pr(|X_{I,n} - \mathsf{CP}(\bar{t})| \leq d) \;\; \geq \;\; 1 - 2e^{-2nd^2},$$

for $d > 0$. Having this inequality in place, it is easy to show that

$$\Pr(|X_{I,n} - \mathsf{CP}(\bar{t})| \leq \epsilon) \;\; \geq \;\; 1-\delta,$$

where $\epsilon > 0$, $0 < \delta < 1$, and $n = 1/2\epsilon^2 \cdot \ln(2/\delta)$. It remains to show that $X_{I,n}$ is computable in polynomial time in the size of I, $1/\epsilon$, and $\log(1/\delta)$, which immediately implies that there exists a polynomial-time randomized approximation with additive error for $\mathsf{OCQA}(\Sigma, M_\Sigma, Q)$, as needed. This is done via the algorithm Sample: make $n = 1/2\epsilon^2 \cdot \ln(2/\delta)$ calls Sample(I), and return the ratio m/n, where m is the number of times Sample(I) outputs 1.

The Algorithm Sample**.** The above argument heavily relies on the fact that we have access to the randomized algorithm Sample, which we now describe. Recall that Sample(I), where $I = (D, \bar{t})$, should run in polynomial time and output a random variable $X_I : \Omega \to \{0,1\}$, and $\Pr(X_I = 1) = \mathsf{CP}(\bar{t})$. The formal definition follows:

(1) $(\varepsilon, \mathbf{P}) := M_\Sigma(D)$
(2) $s := \varepsilon$
(3) **while** $s(D)$ is inconsistent with Σ **do**:
  (a) $N := \{op \mid s \cdot op \in \mathsf{RS}(D,\Sigma), \mathbf{P}(s, s \cdot op) > 0\}$
  (b) Choose $op \in N$ with probability $\mathbf{P}(s, s \cdot op)$
  (c) $s := s \cdot op$
  end while
(4) If $\bar{t} \in Q(s(D))$, then **return** 1; otherwise, **return** 0.

Roughly, the algorithm randomly chooses a (D,Σ)-repairing sequence s and returns 1 if $s(D)$ entails the input tuple $\bar{t}$; otherwise, it returns 0. Note that the while-loop terminates since there are no failing sequences and, by construction, Sample(I) is a random variable over (Ω, π). A key property that the algorithm Sample exploits is the tree-like structure of the repairing Markov chain $M_\Sigma(D)$. This allows the algorithm to construct in polynomial time, in a level-by-level fashion, the probability $\pi(s)$ of a sequence $s \in \mathsf{ras}(M_\Sigma(D))$, where π is the hitting distribution of $M_\Sigma(D)$. We can then show that Sample is indeed the desired randomized algorithm:

Proposition 5.6. *Consider an instance $I = (D, \bar{t})$ of the problem $\mathsf{OCQA}(\Sigma, M_\Sigma, Q)$. The following hold:*

- Sample(I) *terminates after polynomially many steps.*
- $\Pr(\mathsf{Sample}(I) = 1) = \mathsf{CP}(\bar{t})$.

Note that in fact one shows the following:

$$\Pr(\mathsf{Sample}(I) = 1) \;=\; \sum_{(D',p) \in [\![D]\!]_{M_\Sigma} \text{ and } \bar{t} \in Q(D')} p.$$

In general, this may be different from the probability $\mathsf{CP}(\bar{t})$ since the denominator $\sum_{(D',p)\in[\![D]\!]_{M_\Sigma}} p$ is missing. Nevertheless, since M_Σ is non-failing, $\mathsf{CP}(\bar{t})$ is not a conditional probability (i.e., the denominator equals to one), and thus $\Pr(\mathsf{Sample}(I) = 1) = \mathsf{CP}(\bar{t})$.

On implementing the approximation scheme. Theorem 5.5 tells us that we can efficiently approximate OCQA with additive error. How can such a scheme be implemented in practice, when Q is an SQL query? We now outline a possible scheme for the common case of deletion updates and key violations.

The user sets numbers ϵ and δ, and computes the number n of samples from it as $1/2\epsilon^2 \cdot \ln(2/\delta)$. To give an idea of the magnitude of this number, it is not small but not very large either: for $\epsilon = \delta = 0.1$, for example, it is 150, which looks like a reasonable price to pay for approximating an intractable problem). We then do the following n times: from each group of tuples in relation R that violate a key (i.e., tuples with the same value of the key), randomly pick at most one tuple to be left there, and collect others in a relation R_{del}. Then run the original query Q in which each relation R is replaced with $R - R_{del}$, and append the outcome to a temporary table T that collects results of the n random runs. When T is computed, for each tuple $\bar{t}$ we compute the number of times $n_{\bar{t}}$ it occurs with a simple aggregate query, and return $n_{\bar{t}}/n$ for each such $\bar{t}$. This is the approximation of $\mathsf{CP}(\bar{t})$.

The question is how this will perform in practice. The right way to answer this question is to design a proper set of experiments, which we intend to do in the followup work. At this early stage we wanted to make sure that the idea of the approximation is plausible. The biggest obstacle to it would be a significant slowdown of modified queries in which relations R are replaced with $R - R_{del}$; such slowdowns due to optimizer difficulties are not unheard of, even in simple cases of mild changes in queries [13]. Repeating a much slower modified query multiple times would not be feasible. We thus ran a few initial experiments on such modified queries, which showed that their performance is quite similar to that of the original query. This gives us hope that, in addition to its theoretical guarantees, the approach can be successfully used in practice.

6 CONCLUSIONS AND FUTURE WORK

We presented a new framework for consistent query answering that provides us with a much better understanding of how repairs are constructed, and consequently with a new flexible paradigm for query answering. Based on it, we were able to push the boundaries of the CQA approach quite far, providing an efficient approximation scheme for all FO queries in the setting where constraints come from the most commonly used classes, and where arbitrary deletion updates are allowed. These promising early results lead to a host of new questions we would like to address.

Approximation for Insertions and Deletions. How do we extend the additive error approximation scheme to handle both insertions and deletions? In this case we have to approximate a ratio and not a single number.

More Expressive Languages. We would like to extend the approach to other languages with tractable data complexity (e.g., with aggregates [3] or various flavors of Datalog [34]), and extend the class of constraints, e.g., with active constraints that also suggest an update operation as in [11].

Different Types of Updates. We can also consider modifications of tuples. Actually, the approach closest in the spirit to ours [23] looked at such operations.

Equally Likely Repairs. Another idea of [23] that we could apply in our setting is to use the proportion of repairs in which a given tuple is in the answer as a measure of certainty. This means that every repair (not even every repairing sequence) is equally likely.

Preferences. In our framework, we express likelihoods of operations, repairs, and query answering via probabilities. A milder way is preferences, among operations or sequences of operations, similarly to [18, 36], or at the source level, if inconsistent data is the result of integration [15].

Null Values. We could also use nulls (either SQL or marked) in repairs, in cases when we insisted on adding tuples from the base. This, of course, would bring to the fore many issues related to incomplete information handling [33].

Other Data Models. We could also try to use our approach with other data models (tree and graph-structured), as well as probabilistic databases [32].

Optimizations. There are several ideas in the literature that were proposed to speed up CQA, and which appear to be applicable in our framework. Among them is the idea of localization of repairs [17], i.e., concentrating on the part of the database where violations occur, and dealing with partial satisfaction of constraints [10, 31].

Query Rewriting. One can express additive error approximations by means of FO queries. These queries themselves are dependent on the inconsistent database but their size is not. To understand their practical feasibility we need to conduct an experimental study.

Ontological Query Answering. The standard ABC semantics for inconsistent databases has been lifted to the ontological setting giving rise to the so-called ABox repair semantics [30]. As it is computationally hard, even for lightweight ontology languages, several other semantics have been developed with the aim of approximating the set of consistent answers [8, 9, 30]. Approximation in this setting refers to a subset of consistent answers, without giving any guarantees. We would like to apply our approach in this scenario, with the aim of providing probabilistic guarantees.

Acknowledgements. We thank the anonymous referees for their useful feedback. This work was supported by the EPSRC Programme Grant EP/M025268/ "VADA: Value Added Data Systems - Principles and Architecture".

REFERENCES

[1] Serge Abiteboul, T.-H. Hubert Chan, Evgeny Kharlamov, Werner Nutt, and Pierre Senellart. 2011. Capturing continuous data and answering aggregate queries in probabilistic XML. *ACM Trans. Database Syst.* 36, 4 (2011), 25:1–25:45.
[2] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.
[3] Marcelo Arenas, Leopoldo E. Bertossi, Jan Chomicki, Xin He, Vijay Raghavan, and Jeremy P. Spinrad. 2003. Scalar aggregation in inconsistent databases. *Theor. Comput. Sci.* 296, 3 (2003), 405–434.
[4] Sanjeev Aror and Boaz Barak. 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press.

[5] Leopoldo E. Bertossi. 2011. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers.
[6] Leopoldo E. Bertossi and Loreto Bravo. 2017. Consistency and trust in peer data exchange systems. *TPLP* 17, 2 (2017), 148–204.
[7] Leopoldo E. Bertossi and Lechen Li. 2013. Achieving Data Privacy through Secrecy Views and Null-Based Virtual Updates. *IEEE Trans. Knowl. Data Eng.* 25, 5 (2013), 987–1000.
[8] Meghyn Bienvenu. 2012. On the Complexity of Consistent Query Answering in the Presence of Simple Ontologies. In *AAAI*.
[9] Meghyn Bienvenu and Riccardo Rosati. 2013. Tractable Approximations of Consistent Query Answering for Robust Ontology-based Data Access. In *IJCAI*. 775–781.
[10] Andrea Calì, Domenico Lembo, and Riccardo Rosati. 2003. Query rewriting and answering under constraints in data integration systems. In *IJCAI*. 16–21.
[11] Luciano Caroprese, Sergio Greco, and Ester Zumpano. 2009. Active Integrity Constraints for Database Consistency Maintenance. *IEEE Trans. Knowl. Data Eng.* 21, 7 (2009), 1042–1058.
[12] Jan Chomicki and Jerzy Marcinkowski. 2005. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.* 197, 1-2 (2005), 90–121.
[13] Jens Claußen, Alfons Kemper, Guido Moerkotte, Klaus Peithner, and Michael Steinbrunn. 2000. Optimization and Evaluation of Disjunctive Queries. *IEEE Trans. Knowl. Data Eng.* 12, 2 (2000), 238–260.
[14] Nilesh N. Dalvi and Dan Suciu. 2007. Efficient query evaluation on probabilistic databases. *VLDB J.* 16, 4 (2007), 523–544. https://doi.org/10.1007/s00778-006-0004-3
[15] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. 2004. Tackling Inconsistencies in Data Integration through Source Preferences. In *IQIS*. 27–34.
[16] Xin Luna Dong, Laure Berti-Équille, and Divesh Srivastava. 2013. Data Fusion: Resolving Conflicts from Multiple Sources. In *WAIM*. 64–76.
[17] Thomas Eiter, Michael Fink, Gianluigi Greco, and Domenico Lembo. 2008. Repair localization for query answering from inconsistent databases. *ACM Trans. Database Syst.* 33, 2 (2008), 10:1–10:51. https://doi.org/10.1145/1366102.1366107
[18] Ronald Fagin, Benny Kimelfeld, and Phokion G. Kolaitis. 2015. Dichotomies in the Complexity of Preferred Repairs. In *PODS*. 3–15.
[19] Gaëlle Fontaine. 2015. Why Is It Hard to Obtain a Dichotomy for Consistent Query Answering? *ACM Trans. Comput. Log.* 16, 1 (2015), 7:1–7:24.
[20] Ariel Fuxman, Elham Fazli, and Renée J. Miller. 2005. ConQuer: Efficient Management of Inconsistent Databases. In *SIGMOD*. 155–166.
[21] Ariel Fuxman and Renée J. Miller. 2007. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.* 73, 4 (2007), 610–635.
[22] Floris Geerts, Fabian Pijcke, and Jef Wijsen. 2015. First-Order Under-Approximations of Consistent Query Answers. In *SUM*. 354–367.
[23] Sergio Greco and Cristian Molinaro. 2012. Probabilistic query answering over inconsistent databases. *Ann. Math. Artif. Intell.* 64, 2-3 (2012), 185–207.
[24] Wassily Hoeffding. 1963. Probability Inequalities for Sums of Bounded Random Variables. *J. Amer. Statist. Assoc.* 58, 301 (1963), 13–30.
[25] Mark Jerrum. 2003. *Counting, Sampling and Integrating: Algorithms and Complexity*. Birkhäuser Basel.
[26] Richard M. Karp and Richard J. Lipton. 1980. Some Connections between Nonuniform and Uniform Complexity Classes. In *STOC*. 302–309.
[27] John Kemeny and J. Laurie Snell. 1983. *Finite Markov Chains*. Springer.
[28] Paraschos Koutris and Dan Suciu. 2014. A Dichotomy on the Complexity of Consistent Query Answering for Atoms with Simple Keys. In *ICDT*. 165–176.
[29] Paraschos Koutris and Jef Wijsen. 2015. The Data Complexity of Consistent Query Answering for Self-Join-Free Conjunctive Queries Under Primary Key Constraints. In *PODS*. 17–29.
[30] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. 2015. Inconsistency-tolerant query answering in ontology-based data access. *J. Web Sem.* 33 (2015), 3–29.
[31] Nicola Leone et al. 2005. The INFOMIX system for advanced integration of incomplete and inconsistent data. In *SIGMOD*. 915–917.
[32] Xiang Lian, Lei Chen, and Shaoxu Song. 2010. Consistent Query Answers in Inconsistent Probabilistic Databases. In *SIGMOD*. 303–314.
[33] Leonid Libkin. 2014. Incomplete data: what went wrong, and how to fix it. In *PODS*. 1–13.
[34] Thomas Lukasiewicz, Maria Vanina Martinez, Andreas Pieris, and Gerardo I. Simari. 2015. From Classical to Consistent Query Answering under Existential Rules. In *AAAI*. 1546–1552.
[35] Carsten Lutz and Frank Wolter. 2015. On the Relationship between Consistent Query Answering and Constraint Satisfaction Problems. In *ICDT*. 363–379.
[36] Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski. 2012. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.* 64, 2-3 (2012), 209–246.
[37] Seinosuke Toda and Osamu Watanabe. 1992. Polynomial Time 1-Turing Reductions from #PH to #P. *Theor. Comput. Sci.* 100, 1 (1992), 205–221.
[38] Vijai Vazirani. 2001. *Approximation Algorithms*. Springer.
[39] Jef Wijsen. 2005. Database Repairing Using Updates. *ACM Trans. Database Syst.* 30, 3 (Sept. 2005), 722–768.
[40] Jef Wijsen. 2014. A Survey of the Data Complexity of Consistent Query Answering under Key Constraints. In *FoIKS*. 62–78.
[41] Xiaoxin Yin, Jiawei Han, and Philip S. Yu. 2008. Truth Discovery with Multiple Conflicting Information Providers on the Web. *IEEE Trans. on Knowl. and Data Eng.* 20, 6 (2008), 796–808.