On the Decidability of Consistent Query Answering

Marcelo Arenas[1,⋆] and Leopoldo Bertossi[2,⋆⋆]

[1] Pontificia Universidad Católica de Chile, Department of Computer Science, Santiago, Chile. Email: marenas@ing.puc.cl

[2] Carleton University, School of Computer Science, Ottawa, Canada. Email: bertossi@scs.carleton.ca

Abstract. Consistent query answering (CQA) is about formally characterizing and computing semantically correct answers to queries posed to a database that may fail to satisfy certain integrity constraints. In this paper, we revisit the decidability status of consistent query answering by considering different parameters of the problem as input to the decision problem. More specifically, we obtain some new results about the undecidability and combined complexity of CQA.

1 Introduction

Consistent query answering (CQA) is about formally characterizing and computing semantically correct answers to queries posed to a database that may fail to satisfy certain integrity constraints (ICs). This problem was brought into the main stream research on foundations of databases by [3]. Informally, a consistent answer to a query $\mathcal{Q}$ posed to a relational database instance D that may not satisfy a set IC of integrity constraints is as a usual answer to $\mathcal{Q}$ that can be obtained from every *minimal repair* of D. A minimal repair D' of D satisfies IC and minimally departs from D (after enforcing the ICs). Different repair semantics can be obtained by playing with different notions of distance between database instances. In this paper, we stick to the original distance introduced and studied in [3]: A repair of D makes minimal under set inclusion the set of tuples in $(D \smallsetminus D') \cup (D' \smallsetminus D)$, i.e., the symmetric set difference. In particular, this assumes that IC violations are solved by insertions or deletions of whole database tuples. For recent surveys of CQA we refer to [7, 13].

The complexity of CQA problem, i.e., of deciding if a tuple is a consistent answer to a query under a given set of ICs, has been investigated in several papers and under several repairs semantics [11, 12, 27, 18, 8, 21, 28, 2, 23]. As expected most all the complexity results have been about *data complexity*, i.e., in terms of the size of the original instance D. Few results have been presented on combined complexity, where other parameters are also taken into account,

⋆ M. Arenas was supported by FONDECYT grant 1090565.

⋆⋆ Faculty Fellow of the IBM Center for Advanced Studies. Also affiliated to the University of Concepcion, Chile.

e.g. size of the query or of the set of ICs. There are also few results about undecidability of CQA. Exceptions in this direction are [11, 8]; and both refer to a form of *combined undecidability*, i.e., without a fixed query and a fixed set of ICs. We do not elaborate more on [8], because the repair semantics is different from the one investigated in this paper. Furthermore, the results in [8] heavily rely on having denial constraints that involve numerical attributes.

However, some results in [11] are highly relevant in our context. The framework there considers a combination of key constraints (KCs) and inclusion dependencies (IDs). In case of inconsistency, KCs are repaired via tuple deletions; but IDs are repaired only by insertion of tuples. If the original instance is consistent wrt the KCs, then inconsistencies are solved by insertion of whole tuples, which can happen in possibly many different ways, and only constrained by the KCs. The new tuples can take values in the database domain associated to the schema. In this case, the repairs minimize the set of inserted tuples wrt set inclusion.

It is proved in [11] that CQA is undecidable. This result is obtained by reduction from the problem of deciding entailment of dependencies from a set of functional dependencies (FDs) and inclusion dependencies: $\mathcal{F} \cup \mathcal{I} \models \delta$? Here, $\mathcal{F}$ and $\mathcal{I}$ are sets of FDs and IDs, respectively, and δ is a dependency. This problem is undecidable [1, theorem 9.2.4]. A careful examination of the proof in [11, theorems 3.2, 3.4], shows that the reduction in it requires, in essence, generating for the combination of $\mathcal{F}, \mathcal{I}$ and δ an ad hoc combination of KCs, IDs, a query and a database instance. In consequence, the undecidability result involves as input to the problem all the three natural parameters of CQA, namely the ICs, the query and the instance. The set $\mathcal{I}$ is actually a cyclic set of IDs.

In this paper, we revisit the decidability status of consistent query answering by considering different parameters of the problem as input to the decision problem. We obtain some new results about the undecidability and combined complexity of CQA. More specifically, we prove that CQA is undecidable for all the combinations of the possible inputs (instance, ICs, query) to the problem depending upon the parameters that are left fixed. We also establish the decidability of CQA for a broad class of universal ICs, with the combination of instance, ICs, and query as the input. Furthermore, we study the combined complexity on data and ICs (fixed query) for this class of dependencies.

2 Preliminaries

If we start from a database schema $\mathcal{S}$, including a countably infinite database domain U, we may consider the first-order language $L(\mathcal{S})$ based on the predicates in $\mathcal{S}$ and the constants for the elements of U. Database instances for schema $\mathcal{S}$ are Herbrand structures with domain U and finite extensions for the interpretations of the predicates in $\mathcal{S}$. In consequence, a database instance D can be identified with a set of ground atoms $R(\bar{t})$, with $R \in \mathcal{S}$ and $\bar{t}$ a finite sequence of elements of U of the same length as the arity of R. The *active domain* of D, denoted by $dom(D)$, is the finite set that contains all the elements of U that appear in D.

We also allow built-in predicates in $L(\mathcal{S})$, like $=, \neq, <$, etc., that have fixed and predefined extensions in a database instance.

An integrity constraint (IC) over a database schema $\mathcal{S}$ is a first-order sentence in $L(\mathcal{S})$. In particular, a universal IC is a sentence of the form $\forall \bar{x}\, \psi$, where ψ does not contain any quantifiers. We use IC to denote a set of integrity constraints, and we assume that IC is consistent in the sense that there is some database instance that makes it true, but not necessarily by the one at hand. It is always the case that in CQA, and also in this paper, we consider database instances that may not satisfy the given set of ICs.

Definition 1. A database instance D is *consistent* with respect to IC if D satisfies IC, denoted by $D \vDash IC$; and inconsistent, otherwise. □

Definition 2 ([3]).

(a) The *distance* $\Delta(D_1, D_2)$ between database instances D_1, D_2 is the symmetric difference $(D_1 \smallsetminus D_2) \cup (D_2 \smallsetminus D_1)$.
(b) For a fixed instance D and instances D', D'': $D' \leq_D D''$ iff $\Delta(D, D') \subseteq \Delta(D, D'')$.
(c) For a given database instances D, we say that an instance D' is a *repair* of D wrt IC iff $D' \vDash IC$ and D' is $\leq_D$-*minimal* in the class of database instances that satisfy IC. □

Notice that the extensions of the built-in predicates do not contribute to Δ, because they have fixed extensions, identical in every database instance.

Queries are formulas of $L(\mathcal{S})$. If a query is a sentence, i.e., it has not free variables, it is called a *Boolean* query. We assume that queries and ICs satisfy the usual syntactic *safety* conditions [25] that make them *domain independent* [17, 24]. This means that they can be evaluated or checked against the active domain of the database instance.

For a query $\mathcal{Q}(\bar{x})$, with free variables in $\bar{x}$, and a database instance D, a tuple $\bar{t}$ of constants of U is an answer to $\mathcal{Q}$ in D iff $D \models \mathcal{Q}[\bar{t}]$, i.e., $\mathcal{Q}$ is true in D when the variables in $\bar{x}$ are replaced by the corresponding values in $\bar{t}$.

Definition 3 ([3]). Given a set of integrity constraints, a (ground) tuple $\bar{t}$ is a *consistent answer* to a query $\mathcal{Q}(\bar{x})$ in a database instance D wrt a set of integrity constraints IC, denoted $D \models_{IC} \mathcal{Q}[\bar{t}]$, if for every repair D' of D wrt IC, $D' \vDash \mathcal{Q}[\bar{t}]$. If $\mathcal{Q}$ is Boolean, then *yes* is the consistent answer if for every repair D' of D, $D' \vDash \mathcal{Q}$; otherwise the consistent answer is *no*. □

3 The Decision Problem of CQA

The computational problem of retrieving consistent answers to a query from a possibly inconsistent database can be studied by concentrating on the *decision problem of consistent query answering.* Actually, this gives rise to several decision problems depending on which of the natural parameters involved are considered to be fixed and which are considered to be a part of the input. The most basic decision problems of CQA are the following.

Definition 4.

(a) For a Boolean query $\mathcal{Q}$ and a consistent set IC of integrity constraints:

$$\textit{d-CQA}(\mathcal{Q}, IC) := \{D \mid D \text{ is a database instance and } D \models_{IC} \mathcal{Q}\}.$$

(b) For a database instance D and a Boolean query $\mathcal{Q}$:

$$\textit{ic-CQA}(D, \mathcal{Q}) := \{IC \mid IC \text{ is a consistent set of ICs and } D \models_{IC} \mathcal{Q}\}.$$

(b) For a database instance D and a consistent set IC of integrity constraints:

$$\textit{q-CQA}(D, IC) := \{\mathcal{Q} \mid \mathcal{Q} \text{ is a Boolean query and } D \models_{IC} \mathcal{Q}\}.$$

□

Problem $\textit{d-CQA}(\mathcal{Q}, IC)$ is the problem of CQA *in data*, where the ICs and the query are fixed -the *parameters* of the problem- and the database is variable, i.e., the input. The *data complexity* [26] of CQA is precisely the complexity of this problem, for different classes of parameters IC, $\mathcal{Q}$. As common in database research, this is the problem that has been investigated the most. This problem is decidable for all common classes of ICs and queries, and tight complexity bounds have been established [11, 12, 27, 18, 8, 21, 28, 2, 23].

We may consider combined versions of the CQA decision problem. For example, for a given Boolean query $\mathcal{Q}$:

$$\{d, ic\}\textit{-CQA}(\mathcal{Q}) \ := \ \{(D, IC) \mid D \text{ is a database instance, } IC \text{ is a set of ICs and } D \models_{IC} \mathcal{Q}\}, \quad (1)$$

and, more generally,

$$\{d, q, ic\}\textit{-CQA} \ := \ \{(D, \mathcal{Q}, IC) \mid D \text{ is a database instance, } \mathcal{Q} \text{ is a Boolean query, } IC \text{ is a set of ICs and } D \models_{IC} \mathcal{Q}\}. \quad (2)$$

In what follows, when we refer to $\{d\}\textit{-CQA}(\mathcal{Q}, IC)$, we mean $\textit{d-CQA}(\mathcal{Q}, IC)$, as in Definition 4; the same for *ic* and *q*. The complexity of $\{d, q\}\textit{-CQA}(IC)$ is usually called *combined complexity* of CQA (with fixed ICs). It is measured in terms of the size of both the database and the query. In the case of $\{q, ic, d\}\textit{-CQA}$, the instance, the query and the set of ICs become part of the input. All this decision problems can also be stated for queries with free variables, say $\mathcal{Q}(\bar{x})$, and tuples $\bar{t}$ of constants in U, asking if $\bar{t}$ is a consistent answer. Since the schema has the constants in U, we may restrict ourselves to Boolean queries. As already indicated above, it holds:

Proposition 1 ([11]). $\{d, q, ic\}\textit{-CQA}$ is undecidable. □

The proof of this proposition given in [11] uses *key constraints* and cyclic sets of *referential integrity constraints* (RICs) [1]. The existential quantifiers of the RICs are satisfied by picking up values from the underlying database domain

U. We should mention that, in [10], $\{q, ic, d\}$-*CQA* was made decidable for this type of ICs by considering repairs based on introduction of null values as used in SQL databases. Those null values are used for the existential variables in the RICs. Actually, the decidability follows in [10] from the use of disjunctive Datalog programs with stable model semantics [19, 15] to specify the repairs of the original instance wrt universal ICs and RICs, including those considered in [11]. These *repair programs* have been used before [5, 20, 6], and they provide an upper bound of Π_2^P for the time complexity of *d-CQA*$(\mathcal{Q}, IC)$ [14]. This is also a lower bound since CQA can be Π_2^P-complete in data [12].

4 Undecidability of CQA

Now we turn to some of the other decision problems for CQA. Clearly, for $Y \subseteq X \subseteq \{d, ic, q\}$, if *X-CQA* is decidable, then also *Y-CQA* is decidable, but not necessarily the other way around. We first consider the problem of CQA *in integrity constrains.*

Proposition 2. There exist a database instance D and a Boolean query $\mathcal{Q}$ such that *ic-CQA*$(D, \mathcal{Q})$ is undecidable.

Proof. Let $\mathcal{S}$ be a database schema that contains a binary predicate E and a unary predicate P. Consider the instance D of $\mathcal{S}$ whose extension for P is $\{P(a)\}$, where a is an element of U, and whose extension for E is $\emptyset$. Furthermore, consider the Boolean query $\mathcal{Q}: \ \neg P(a)$; and the problem $SAT := \{\psi \in L(\{E\}) \mid \psi \text{ is satisfiable}\}$. This problem is undecidable over finite graphs [16, sec. 7.2.1], and can be reduced to the complement of *ic-CQA*$(D, \mathcal{Q})$ as follows: Given a first-order sentence $\psi \in L(\{E\})$, define a set of integrity constraints IC by $\{\psi \leftrightarrow \exists x P(x)\}$. It is easy to check that $\psi \in SAT$ iff $D \not\models_{IC} \mathcal{Q}$. □

Now we address the problem of CQA *in queries*, that is, when queries are the input of the problem. The complexity of this decision problem is the *query complexity* of CQA [26].

Proposition 3. There exists a database instance D and a consistent set IC of integrity constraints such that *q-CQA*(D, IC) is undecidable.

Proof. Let $\mathcal{S}$ be a database schema containing a unary predicate P, a binary predicate E, a ternary predicate F and built-in predicate $\leq$ that is interpreted in such a way that $(U, \leq)$ is isomorphic to $(\mathbb{N}, \leq)$. Then let D be the empty instance of $\mathcal{S}$, and IC be a set consisting of the following integrity constraints:

$$\begin{aligned}
&\exists x P(x),\\
&\forall u \forall v (E(u, v) \leftrightarrow \exists y F(u, v, y)),\\
&\forall x \forall y (P(x) \wedge y \leq x \rightarrow \exists u \exists v F(u, v, y)).
\end{aligned}$$

Consider $SAT := \{\psi \in L(\{E\}) \mid \psi \text{ is satisfiable}\}$. As mentioned in the proof of Proposition 2, this problem is undecidable over finite graphs. Next we show that the restriction of SAT to finite graphs can be reduced to the complement of q-$CQA(D, IC)$. In fact, we prove that $\psi \in SAT$ iff $\neg\psi \notin q$-$CQA(D, IC)$.

Assume that $\psi \in SAT$, and let D' be a database instance over $\{E\}$ satisfying ψ. Define a database instance $D^\star$ over $\mathcal{S}$ as follows. The interpretation of E in $D^\star$ coincides with the interpretation of E in D'. Let $\bar{t}_1, \ldots, \bar{t}_m$ be the tuples in the interpretation of E in D', and assume that $a_1, \ldots, a_m$ are the first m elements of U according to linear order $\leq$. Then the interpretations of P and F in $D^\star$ are $\{a_m\}$ and $\{(\bar{t}_i, a_i) \mid 1 \leq i \leq m\}$, respectively. It is straightforward to prove that $D^\star \models IC$. Furthermore, $D^\star$ is a repair of D since no proper subset of $D^\star$ satisfies IC. Thus, given that $D^\star \models \psi$, we conclude that $D \not\models_{IC} \neg\psi$ and, therefore, $\neg\psi \notin q$-$CQA(D, IC)$. The other implication is immediate, which concludes the proof of the proposition. □

Finally, we also consider the case when integrity constraints and queries are considered to be fixed.

Proposition 4. There exist a Boolean query $\mathcal{Q}$ and a consistent set IC of integrity constraints such that d-$CQA(\mathcal{Q}, IC)$ is undecidable.

Proof. We will reduce to the complement of our problem the halting problem for deterministic Turing machines with empty string on an infinite unidirectional tape with alphabet 0, 1, B (blank). The integrity constraints will codify the dynamics of the machine, and the database instance will contain the transition function δ of the machine. Repairing the instance wrt the dynamics of the machine corresponds to making the machine compute. The initial and final states are q_0, q_f, respectively. We need the following predicates:

1. $Head(t, p)$: At time t the head is at position with number p of the tape.
2. $State(t, q)$: At time t the state is q.
3. $Conf_a(t, p)$: At time t and position p the symbol read is a. Here, $a \neq \mathtt{B}$; and a blank is represented by the absence of 0 or 1.
4. $\delta_{a,a',R}(q, q')$: Being at state q, reading a causes, according to the transition function δ, to write a', move to the right (R) and jump to state q'. There are similar predicates, L, for left.
5. $Succ(n, n')$: For natural numbers n, n', it holds that n' is the successor of n. This predicate can be defined using the built-in predicate $<$, and its definition can be part of IC. Thus, we also assume that the database schema contains a built-in predicate $<$ that is interpreted in such a way that $(U, <)$ is isomorphic to $(\mathbb{N}, <)$.

Assume that a_0 is the first element of U according to linear order $<$. Then IC contains the following sentences:

(a) $Head(a_0, a_0)$ and $State(a_0, q_0)$.
(b) $\forall p(\neg Conf_0(a_0, p) \wedge \neg Conf_1(a_0, p))$. This formula states that that we start with a blank tape. In general, we represent with $\neg Conf_0(t, p) \wedge \neg Conf_1(t, p)$ the fact that at time t in position p there is a blank.

(c) For $a \neq \mathtt{B}$ and $a' \neq \mathtt{B}$:

$$\forall t \forall t' \forall q \forall q' \forall p \forall p' \Big(State(t,q) \wedge Head(t,p) \wedge Conf_a(t,p) \wedge \delta_{a,a',L}(q,q') \wedge Succ(t,t') \wedge Succ(p',p) \ \rightarrow Conf_{a'}(t',p) \wedge Head(t',p') \wedge State(t',q') \Big).$$

There is a similar sentence for $\delta_{a,a',R}$. If a is $\mathtt{B}$, in the previous formula $Conf_a(t,p)$ is replaced by $\neg Conf_0(t,p) \wedge \neg Conf_1(t,p)$, obtaining:

$$\forall t \forall t' \forall q \forall q' \forall p \forall p' \Big(State(t,q) \wedge Head(t,p) \wedge \neg Conf_0(t,p) \wedge \neg Conf_1(t,p) \wedge \delta_{\mathtt{B},a',L}(q,q') \wedge Succ(t,t') \wedge Succ(p',p) \ \rightarrow Conf_{a'}(t',p) \wedge Head(t',p') \wedge State(t',q') \Big).$$

Similar formulas are obtained when $a' = \mathtt{B}$.

(d) For every $a \neq \mathtt{B}$:

$$\forall t \forall t' \forall p \forall p' (Head(t,p) \wedge p \neq p' \wedge Succ(t,t') \rightarrow (Conf_a(t,p') \leftrightarrow Conf_a(t',p'))).$$

The query $\mathcal{Q}$ is $\neg \exists t \, State(t, q_f)$. All these elements determine a particular decision problem of the form $q\text{-}CQA(IC, \mathcal{Q})$, parameterized by IC and $\mathcal{Q}$. Now, we show that this problem is undecidable by reduction from the halting problem.

For a machine M, the database instance $D(M)$ corresponds to the transition function δ by means of the δ-predicates above and their contents. The other relations are empty. It holds that M halts at state q_f iff $D(M) \not\models_{IC} \mathcal{Q}$. In fact:

($\Rightarrow$) In this case, there is a repair $D(M)'$ corresponding to the finite computation of M, i.e., it contains all the configuration tuples that lead to the final state. For this repair, it holds that $D(M)' \not\models \mathcal{Q}$. Thus, $D(M) \not\models_{IC} \mathcal{Q}$.

($\Leftarrow$) Assume M does not halt at state q_f. We have two cases: First, assume that M does not halt (at any state). The infinite computation does not generate a repair, because repairs have finite relations. In this case, repairs can be obtained by deletion of tuples from $D(M)$, and then they correspond to sub-function of the original δ. None of the computations related to such a repair can reach state q_f, because they represent subcomputation of the original machine (the determinism of the machine is crucial here). In this case, all the repairs satisfy $\mathcal{Q}$ and, therefore $D(M) \models_{IC} \mathcal{Q}$. In the second case, M halts at a state different from q_f. A similar argument as in the previous case can be applied. $\square$

It is important to notice that the proof of Proposition 4 relies on the availability of a built-in linear order. Also, in this proof we use universal ICs except for the definition of the successor function. However, this definition, namely $\forall m \forall n (Succ(m,n) \ \leftrightarrow \ m < n \wedge \neg \exists j (m < j \wedge j < n))$, as an IC does not contain "repairable" predicates, i.e., database predicates. Thus, one could also rely on the availability of a built-in successor predicate to prove this proposition, avoiding the use of non-universal constraints.

The propositions above have been established for the *single input cases* in Definition 4. However, several possible inputs can be combined, as in (1) and (2). From Propositions 2, 3 and 4, we obtain:

Theorem 1 (Undecidability of consistent query answering). For every nonempty subset X of $\{d, q, ic\}$, there are cases where X-*CQA* is undecidable. The cases depend upon the parameters of the problem, i.e., on $\{d, q, ic\} \smallsetminus X$. $\Box$

5 Combined Decidability and Complexity of CQA

In this section, we concentrate on a particular but common class of ICs.

Definition 5. An integrity constraint is a *safe universal IC* if both:

(a) It is logically equivalent to a sentence of the form

$$\forall(\bigvee_{i=1}^{m} P_i(\bar{x}_i) \vee \bigvee_{j=1}^{n} \neg Q_j(\bar{y}_j) \vee \psi), \tag{3}$$

where $\forall$ represents the universal closure of the formula, $\bar{x}_i$, $\bar{y}_j$ are tuples of variables, the P_i, Q_j are database predicates, and ψ is a formula that mentions only the built-in predicates.

(b) Each variable in an $\bar{x}_i$ or ψ in (3) appears in some $\bar{y}_j$. $\Box$

The second condition is the safety condition that guarantees domain independence [17, 24]. It is easy to see that, according to this definition, the ICs in Proposition 4 are not safe, nor domain independent.

Theorem 2. For finite sets IC of safe universal ICs, and Boolean queries $\mathcal{Q}$, the problem of CQA, i.e., $\{(D, \mathcal{Q}, IC) \mid D \models_{IC} \mathcal{Q}\}$, is decidable.

Proof. We may assume that the sentences in IC are in the clausal form (3). This representation for ICs allows us to determine all the repairs of a database that does not satisfy them: From the domain independence condition, the finitely many tuples that participate in a violation and whose modification may lead to a restoration of the consistency of the database, can be obtained by posing to D, for each IC (3), the domain independent query about the set of violating tuples:

$$V(\bar{y}_1, \ldots, \bar{y}_n) \coloneq \bigwedge_{j=1}^{n} Q_j(\bar{y}_j) \wedge \neg\psi \wedge \bigwedge_{i=1}^{m} \neg P_i(\bar{x}_i).$$

Repairs are obtained by deleting Q_j-ground tuples and/or inserting P_i-ground tuples. In this way, the finitely many repairs can be computed. Then, the query $\mathcal{Q}$ can be evaluated in every single repair. If for all of them it becomes true, the answer is *yes*, otherwise, is *no*. $\Box$

Notice that the decision algorithm in the proof of Theorem 2 requires the computation all of possible repairs; and we know that there may be exponentially many of them in the size of the database [4].

This decidability result extends similar implicit results [5, 20, 6, 10] that guarantee decidability for certain syntactic classes of universal ICs. Their syntax makes them safe, and then, also domain independent. This is obtained from the representation of repairs as stable models of disjunctive logic programs. A direct proof of decidability of $\{d, ic, q\}$-CQA for this class of universal constraints plus RICs can be found in [10] (existential variables are satisfied with an SQL null). Decidability of $\{d, ic, q\}$-CQA is proved in [11] for sets of *non-conflicting* key constrains and inclusion dependencies.

In the following, we study a form of *combined* complexity of CQA, where the input consists of the database and the ICs.[3] This is a special case of the general $\{d, ic\}$-CQA problem in (1) where only universal, domain independent ICs are considered in the input.

Theorem 3.

(a) For every Boolean query $\mathcal{Q}$,

$$CQA(\mathcal{Q}) := \{(D, \varphi) \mid \varphi \text{ is a safe universal IC and } D \models_{\{\varphi\}} \mathcal{Q}\}$$

is in *co-2-NEXP*.

(b) There are Boolean queries for which the same problem is *co-NEXP*-hard.

Proof. (a) We prove that the complement of $CQA(\mathcal{Q})$ belongs to *2-NEXP*, that is, we prove that there exists a non-deterministic Turing machine M that works in double exponential time and accepts the complement of $CQA(\mathcal{Q})$. Let D be a database instance and φ a safe universal IC. In order to verify whether $(D, \varphi) \notin CQA(\mathcal{Q})$, Turing machine M first transforms φ into an equivalent IC ψ of the form (3), then it guesses a repair D' of D wrt ψ, and finally it checks whether $D' \not\models \mathcal{Q}$ (which implies that $D \not\models_{\{\varphi\}} \mathcal{Q}$). Sentence ψ can be of exponential size in the size of φ, and it takes exponential time to generate this formula. Checking that D' is a repair of D can be done in exponential time in the size of ψ and D, because it might be necessary to compare D' for inclusion with exponentially many candidates (see the proof of Theorem 2). Finally, verifying that $D' \not\models \mathcal{Q}$ can be done in polynomial time in the size of D' since $\mathcal{Q}$ is assumed to be a fixed query. Thus, we conclude that M works in double exponential time in the size of φ and D.

(b) Let P be a unary predicate and $a \in U$. Next we prove that the complement of $CQA(\mathcal{Q})$ is *NEXP*-hard, where Q is the Boolean query $P(a)$. More precisely, next we provide a reduction from SAT for the Bernays-Schoenfinkel's syntactic class of first-order formulas (BSC) to the complement of $CQA(\mathcal{Q})$. The former problem is decidable and *NEXP*-complete [22, chapter 20].

[3] Other cases of combined complexity for CQA around key constraints and inclusion dependencies can be found in [11].

Consider a schema $\mathcal{S}$, and let χ: $\exists x_1 \cdots \exists x_n \forall \bar{y}\, \psi(x_1, \ldots, x_n, \bar{y}) \in L(\mathcal{S})$, with ψ quantifier free, be an instance for *BSC*. Without loss of generality, assume that $\mathcal{S}$ does not contain unary predicate P. If χ is satisfiable, then it is satisfiable in the universe $U = \{1, \ldots, n\}$. In consequence, a sentence in *BSC* is satisfiable iff it is satisfiable in a finite structure (actually with finite universe), which allows us to use this result in our context.

Let $a_1, \ldots, a_n$ be pairwise distinct elements from U, which are all distinct from a, $U_1, \ldots, U_n$ be unary predicates not contained in $\mathcal{S} \cup \{P\}$, and D be the empty database instance over $\mathcal{S} \cup \{U_1, \ldots, U_n, P\}$. Furthermore, let φ be the conjunction of:

$$\bigwedge_{i=1}^{n} \forall x \forall y \left(U_i(x) \wedge U_i(y) \rightarrow x = y \right), \tag{4}$$

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} U_i(a_j), \tag{5}$$

$$\forall x_1 \cdots \forall x_n \left(U_1(x_1) \wedge \cdots \wedge U_n(x_n) \rightarrow \forall \bar{y}\, \psi(x_1, \ldots, x_n, \bar{y}) \right) \vee P(a). \tag{6}$$

The first two sentences make sure that each U_i contains exactly one element of U. If χ is satisfiable, then there will be a repair where each of the $U_1, ..., U_n$ will contain exactly one element of U and nothing else, and P will be empty. In that repair, the query $P(a)$ will be false and, thus, $D \not\models_{\{\varphi\}} P(a)$. On the other hand, if χ is not satisfiable, then no matter how $U_1, \ldots, U_n$ and the relations in $\mathcal{S}$ are populated, sentence

$$\forall x_1 \cdots \forall x_n \left(U_1(x_1) \wedge \cdots \wedge U_n(x_n) \rightarrow \forall \bar{y}\, \psi(x_1, \ldots, x_n, \bar{y}) \right)$$

will not be satisfied. Therefore, all the repairs of D will include element a into P in order to satisfy (6), which implies that $D \models_{\{\varphi\}} P(a)$. Hence, we conclude that χ is satisfiable iff $D \not\models_{\{\varphi\}} P(a)$.

The IC φ is clearly universal, but we have to check safety. The first two conjuncts obviously are. For the third conjunct, due to the universal quantification on $x_1, ..., x_n$ and their occurrence in the predicates $U_1, ..., U_n$, the only source of non-safety could be formula ψ. There is no reason for ψ to be safe. However, for the purpose of establishing our complexity lower-bound, we may restrict ourselves to a subclass of sentences in *BSC* that still is *NEXP*-hard, namely sentences that encode *bounded* tiling problems. Those sentences turn out to be safe (cf. [9, theorem 6.2.21] for details).[4] $\square$

[4] In particular, due to the bounded domain to be tiled, for each instance we do not need to go beyond a finite numerical domain U. The same would happen if we were encoding computations of Turing machines that are bounded in time.

References

[1] Abiteboul, S., Hull, R. and Vianu, V. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Afrati, F. and Kolaitis, Ph. Repair Checking in Inconsistent Databases: Algorithms and Complexity. *Proc. of the International Conference on Database Theory (ICDT 09)*, 2009, pp. 31-41.

[3] Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. *Proc. ACM Symposium on Principles of Database Systems (PODS 99)*, 1999, pp. 68-79.

[4] Arenas, M., Bertossi, L., Chomicki, J., He, X., Raghavan, V. and Spinrad, J. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science*, 2003, 296(3): 405-434.

[5] Arenas, M., Bertossi, L. and Chomicki, L. Answer Sets for Consistent Query Answering in Inconsistent Databases. *Theory and Practice of Logic Programming*, 2003, 3(4-5):393-424.

[6] Barcelo, P., Bertossi, L. and Bravo, L. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. Chapter in *Semantics of Databases*, Springer LNCS 2582, 2003, pp. 1-27.

[7] Bertossi, L. Consistent Query Answering in Databases. In *ACM Sigmod Record*, 2006, 35(2):68-76.

[8] Bertossi, L., Bravo,L., Franconi, E. and Lopatenko, A. The Complexity and Approximation of Fixing Numerical Attributes in Databases Under Integrity Constraints. *Information Systems*, 2008, 33(4):407-434.

[9] Börger, E., Grädel, E. and Gurevich, Y. *The Classical Decision Problem.* Springer, 1997.

[10] Bravo, L. and Bertossi, L. Semantically Correct Query Answers in the Presence of Null Values. *Proc. EDBT International Workshop on Inconsistency and Incompleteness in Databases (IIDB 06).* Springer LNCS 4254, 2006, pp. 336-357.

[11] Cali, A., Lembo, D. and Rosati, R. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. *Proc. ACM Symposium on Principles of Database Systems (PODS 03)*, 2003, pp. 260-271.

[12] Chomicki, J. and Marcinkowski, J. Minimal-Change Integrity Maintenance Using Tuple Deletions. *Information and Computation*, 2005, 197(1-2):90-121.

[13] Chomicki, J. Consistent Query Answering: Five Easy Pieces. *Proc. International Conference on Database Theory (ICDT 07)*, Springer LNCS 4353, 2007, pp. 1-17.

[14] Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. Complexity And Expressive Power Of Logic Programming. *ACM Computer Surveys*, 2001, 33(3):374-425.

[15] Eiter, T., Gottlob, G. and Mannila, H. Disjunctive Datalog. *ACM Transactions on Database Systems*, 1997, 22(3):364-418.

[16] Ebbinghaus, H.-D. and Flum, J. *Finite Model Theory.* 2nd edition, Springer, 1999.

[17] Fagin, R. Horn Clauses and Database Dependencies. *Journal of the ACM*, 1982, 29(4):952–985.

[18] Fuxman, A. and Miller, R. First-Order Query Rewriting for Inconsistent Databases. *Journal of Computer and System Sciences*, 2007, 73(4):610-635.

[19] Gelfond, M. and Lifschitz, V. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9:365-385.

[20] Greco, G., Greco, S. and Zumpano, E. A Logical Framework for Querying and Repairing Inconsistent Databases. *IEEE Transactions on Knowledge and Data Engineering*, 2003, 15(6):1389-1408.

[21] Lopatenko, A. and Bertossi, L. Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics. *Proceedings of the International Conference of Database Theory (ICDT 07)*, Springer LNCS 4353, 2007, pp. 179-193.

[22] Papadimitriou, Ch. *Computational Complexity*. Addison-Wesley, 1994.

[23] Staworko, S. and Chomicki, J. Consistent Query Answering in the Presence of Universal Constraints. *Information Systems*, 2010, 35(1):1-22.

[24] Ullman, J. *Principles of Database and Knowledge-Base Systems, Vol. I.* Computer Science Press, 1988.

[25] Van Gelder, A. and Topor, R. Safety and Correct Translation of Relational Calculus Formulas. *Proceedings of the Sixth ACM Symposium on Principles of Database Systems (PODS 87)*. ACM Press, 1987, pp. 313-327.

[26] Vardi, M. The Complexity of Relational Query Languages. *Proc. ACM Symposium on Theory of Computing (STOC 82)*, 1982, pp. 137-146.

[27] Wijsen, J. Database Repairing Using Updates. *ACM Transactions on Database Systems*, 2005, 30(3):722-768.

[28] Wijsen, J. On the Consistent Rewriting of Conjunctive Queries Under Primary Key Constraints. *Information Systems*, 2009, 34(7):578-601.