Complexity of Repair Checking and Consistent Query Answering

Sebastian Arming[1], Reinhard Pichler[2], and Emanuel Sallinger[3]

1 **University of Salzburg, Salzburg, Austria**
2 **TU Wien, Vienna, Austria**
3 **University of Oxford, Oxford, UK**

Abstract

Inconsistent databases (i.e., databases violating some given set of integrity constraints) may arise in many applications such as, for instance, data integration. Hence, the handling of inconsistent data has evolved as an active field of research. In this paper, we consider two fundamental problems in this context: Repair Checking (RC) and Consistent Query Answering (CQA).

So far, these problems have been mainly studied from the point of view of data complexity (where all parts of the input except for the database are considered as fixed). While for some kinds of integrity constraints, also combined complexity (where all parts of the input are allowed to vary) has been considered, for several other kinds of integrity constraints, combined complexity has been left unexplored. Moreover, a more detailed analysis (keeping other parts of the input fixed – e.g., the constraints only) is completely missing.

The goal of our work is a thorough analysis of the complexity of the RC and CQA problems. Our contribution is a complete picture of the complexity of these problems for a wide range of integrity constraints. Our analysis thus allows us to get a better understanding of the true sources of complexity.

1998 ACM Subject Classification H.2.0 Database Management – General

Keywords and phrases inconsistency, consistent query answering, complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.21

1 Introduction

Database management systems (DBMS) allow the definition of several forms of integrity constraints (ICs) to specify restrictions on the data to be stored. The DBMS ensures that the stored data indeed satisfies the ICs. However, in modern applications where data is integrated from several sources, violations of the ICs may arise even if the data in each single source satisfies the ICs. Hence, the handling of *inconsistent* data (i.e., data violating the given ICs) has evolved as an active field of research, see e.g., [5, 6, 9, 23] for surveys and [11, 13, 16, 18, 20] for recent work. The foundations of this research were laid by Arenas et al. in [4], where the key concepts of *repairs* and of *consistent answers* were introduced.

Given a set C of ICs and a (presumably inconsistent) database instance D, a *repair* of D w.r.t. C is a database instance I which satisfies C and which *differs minimally* from D. Difference and minimality can be defined in several ways. We follow the approach of [4] where repairs are obtained from the original database by the insertion and deletion of tuples and minimality means that the symmetric set difference $\Delta(D, I)$ is minimal w.r.t. subset inclusion. More formally, let $\Delta(D, I) = (D \setminus I) \cup (I \setminus D)$. Then I is a repair of D w.r.t. C if I satisfies C and there does not exist an instance I' that satisfies C and $\Delta(D, I') \subsetneq \Delta(D, I)$.

19th International Conference on Database Theory (ICDT 2016).
Editors: Wim Martens and Thomas Zeume; Article No. 21; pp. 21:1–21:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The idea of *consistent query answers* is that even from an inconsistent database instance D, we can derive consistent information, namely those answers to a query that one would obtain over every repair I of D. More precisely, the set of consistent answers to a query Q for a given database D and ICs C is defined as $\bigcap\{Q(I) \mid I \text{ is a repair of } D \text{ w.r.t. } C\}$.

▶ **Example 1.** Consider the set of constraints $C = \{\mathsf{Course}(p, c) \to \mathsf{Prof}(p)\}$ consisting of a single inclusion dependency which states that every course must be taught by a professor. Now let us define our database $D = \{\mathsf{Course}(\mathsf{db}, \mathsf{alice}), \mathsf{Course}(\mathsf{dm}, \mathsf{bob}), \mathsf{Prof}(\mathsf{alice})\}$ that describes two courses: The course database systems (db) taught by Alice (alice) and discrete mathematics (dm) taught by Bob (bob). We observe that D is inconsistent w.r.t. C: While for the course offered by Alice there exists a corresponding tuple $\mathsf{Prof}(\mathsf{alice})$, there is no such tuple for Bob, violating the single constraint contained in C.

There are two possible *repairs*, that is, consistent databases that differ minimally from D: We can either add Bob as a professor, yielding $I_1 = D \cup \{\mathsf{Prof}(\mathsf{bob})\}$ or we remove the discrete mathematics course, yielding $I_2 = D \setminus \{\mathsf{Course}(\mathsf{dm}, \mathsf{bob})\}$. It is easy to see that I_1 and I_2 are consistent. Yet to check that these are indeed repairs, we have to show that they *differ minimally* from D in terms of symmetric difference and subset minimality. Indeed, both have just a single tuple in their symmetric difference with D, thus there can be no other consistent database instance with a smaller symmetric difference to D. In contrast, the instance $I_3 = \emptyset$ is not a repair as, while it fulfills all constraints, repair I_2 has a smaller – in terms of set inclusion – symmetric difference (I_2 removes only the offending tuple, while I_3 removes all tuples).

Let us now proceed to answering queries. Assume that we pose the query $Q = \mathsf{Course}(p, c)$. The only *consistent answer* to Q given C and D is the tuple $\mathsf{Course}(\mathsf{db}, \mathsf{alice})$ as it is contained in all repairs (both in I_1 and in I_2). In contrast, $\mathsf{Course}(\mathsf{dm}, \mathsf{bob})$ is not a consistent answer, as it is not contained in the repair I_2.

The following decision problems are crucial to deal with inconsistent data:

Repair Checking (RC)
Instance: Databases D and I and a set of constraints C
Question: Is I a repair of D w.r.t. C?

Consistent Query Answering (CQA)
Instance: A database D, a set of constraints C, and a Boolean query Q
Question: Is Q true in all repairs of D w.r.t. C?

Goal. Most research on the RC and CQA problems has focused on data complexity. For the CQA problem, this means that constraints C and query Q are considered as fixed and only the database D is allowed to vary. There are a few exceptions, as some works also consider the combined complexity, where all three parts of the input are allowed to vary, e.g. [8, 3]. However, for several kinds of integrity constraints, the combined complexity has been left unexplored. Moreover, a more detailed analysis (keeping other parts of the input fixed) is completely missing. For instance, what happens if we just fix the integrity constraints C, which is a relatively typical setting in a system where data and queries vary, but constraints stay the same? What about other types of complexity – after all, with three parts of the input, there is a total of seven reasonable combinations.

The goal of our work is a thorough analysis of the complexity of the CQA and RC problems. As an important special case, we also consider the complexity of these problems when the arity of the relation symbols is bounded by a constant. Known results in the area provide important parts of the picture (in particular with respect to data complexity). Yet when considering all possible types of complexity (i.e., parts of the input to be fixed or varying), it turns out that for most cases the exact complexity is actually not known. In this work, we complete the picture, allowing us to get a better understanding of the true sources of complexity.

As far as the queries Q are concerned, we concentrate on the fundamental class of Boolean conjunctive queries. It can be easily verified that all of our results carry over to arbitrary conjunctive queries (i.e., asking if a given tuple is contained in the answer to Q over every repair I) and unions of conjunctive queries. We consider a number of different languages from which the constraints C are taken. The languages considered here range from first-order logic as the most powerful one to inclusion dependencies and key dependencies which are among the least expressive ones. In total, the contribution of this work is a complete picture of the complexity of the RC and CQA problems for a wide range of constraint classes.

Organization. In Section 2 (preliminaries), we introduce the constraint classes and complexity classes that are considered in this work. This will allow us to give an overview of our results in Section 3 – starting with the CQA problem (Section 3.1) and continuing with the RC problem (Section 3.2). The intuition of the results and selected proofs are given in Section 4 for repair checking and after that, in Section 5, for consistent query answering. We give concluding remarks in Section 6.

2 Preliminaries

We assume familiarity with the relational data model [1]. Below we recall some basic notions to fix the notation. A *schema* $\mathcal{S}$ is a triple $(\mathcal{U}, \mathcal{R}, \mathcal{B})$ where $\mathcal{U}$ is a countable domain, $\mathcal{R}$ is a finite set of relation symbols (each with some arity), and $\mathcal{B}$ is a finite set of built-in predicates, e.g. $\mathcal{B} = \{\leq, =\}$. Each built-in predicate from $\mathcal{B}$ comes with some fixed, not necessarily finite, relation over $\mathcal{U}$. We restrict ourselves here to the equality predicate, i.e., $\mathcal{B} = \{=\}$. However, it is easy to see that allowing other standard comparisons does not change our results. The relation symbols of $\mathcal{R}$ and $\mathcal{B}$ are called the *vocabulary* of the schema, and give rise to a language of first-order predicate logic. The *arity* of the schema is the maximum of the arities of the symbols in the vocabulary. A *database instance* is a finite set of *facts* of the form $R(a_1, \ldots, a_n)$ where $R \in \mathcal{R}$ is a relational symbol of arity n, and $a_1, \ldots, a_n$ are all elements from $\mathcal{U}$. Each database instance corresponds to a structure of the vocabulary.

We have already defined the RC and CQA problems in the introduction. We now give additional notation that will be helpful in the sequel. Given a database instance D and a first-order sentence φ such that φ is true in D, i.e. it is true in the structure corresponding to D, we write $D \models \varphi$ and say that D is *consistent* with φ. We often extend this notation to finite sets of first-order sentences Φ, writing $D \models \Phi$ if $D \models \varphi$ for every $\varphi \in \Phi$.

For an arbitrary database instance D, we define the partial order $\leq_D$ on database instances as $I \leq_D I'$ iff $\Delta(D, I) \subseteq \Delta(D, I')$. Given a set of first-order sentences C and a database instance I, we can thus say that I is a *repair* of D w.r.t. C iff $I \models C$ and there is no I' with $I' \lneq_D I$ and $I' \models C$. A *Boolean query* Q is a first-order sentence. We write $D \models_C Q$ to mean that Q is true in all repairs of D w.r.t. C. In this work, we restrict ourselves to Boolean conjunctive queries.

IC languages. Figure 1 shows the hierarchy of the IC languages considered here.

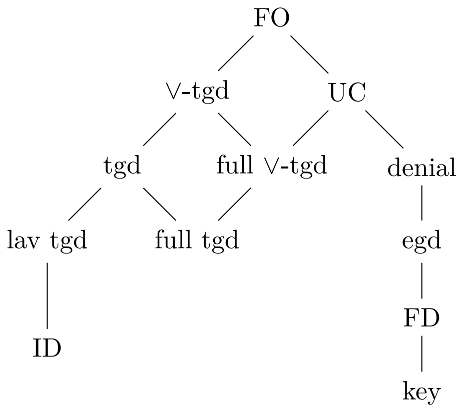

Figure 1 Hierarchy of IC languages.

Besides *domain independent first-order (FO)* sentences, all studied languages arise from restrictions on formulas of the following form

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

where φ, ψ_i are conjunctions of database atoms and β is a quantifier-free formula using only built-in predicates (i.e. equality and – in case of negated form – inequality, in this work) . To ensure safety, we require that every variable in $\vec{x}$ must occur in some relational atom in φ. We assume that constraints do not contain constants.

We call such a constraint a *full* or a *universal constraint (UC)* if it contains no existential quantifiers. A *disjunctive tuple generating dependency (∨-tgd)* has empty β, while an ordinary *tuple generating dependency (tgd)* additionally has $n = 1$. A *local-as-view (lav) tgd* is a tgd where φ is a single atom, and an *inclusion dependency (ID)* is a lav tgd where also ψ_1 is a single atom. A *denial constraint* is of the form $\forall \vec{x} \neg(\varphi(\vec{x}) \wedge \beta(\vec{x}))$ and can be thought of as a universal constraint with empty right hand side. An *equality generating dependency (egd)* is of the form $\forall \vec{x}(\varphi(\vec{x}) \rightarrow x_i = x_j)$ and can be thought of as a denial constraint where β is a single inequality. Given a relation symbol R of arity n, two sets $I \subseteq \{1, \dots, n\}$, $J \subseteq \{1, \dots, n\}$ and pairwise distinct variables $x_1, \dots, x_n, y_1, \dots, y_n$, a *functional dependency (FD)* over R is a formula of the form

$$\forall \vec{x}(R(x_1, \dots, x_n) \wedge R(y_1, \dots, y_n) \wedge \bigwedge_{\alpha \in I} x_\alpha = y_\alpha \rightarrow \bigwedge_{\beta \in J} x_\beta = y_\beta.$$

W.l.o.g., we may exclude trivial FDs by assuming that $I \cap J = \emptyset$ holds. Clearly, FDs are a special case of egds, since we can of course propagate the equalities on the left-hand side into the R-atoms and we can get rid of the conjunction on the right-hand side by splitting such a formula into $|J|$ formulas with identical left-hand side and a single equality on the right-hand side. A *key constraint* is an FD where $I \cup J = \{1, \dots, n\}$ holds.

Note that in this work, we do not distinguish between individual constraints and sets of constraints. In particular, as argued above, an FD or a key constraint corresponds to a set of egds. This has to be kept in mind for the inclusions shown in Figure 1.

Complexity classes. Apart from the more familiar complexity classes P, NP, PSPACE and EXP we will refer to the following classes. First recall that Σ_2P is the class of problems decidable by an NP Turing machine with an NP oracle, and Σ_3P contains the problems decidable by an NP Turing machine with a Σ_2P oracle. A typical complete problem for Σ_3P is the problem $\exists$QSAT$_3$, which asks whether a given quantified boolean formula with three alternating blocks of quantifiers, starting with an existential quantifier, is satisfiable. The co-classes of Σ_2P and Σ_3P are called Π_2P and Π_3P, respectively. In a similar way, Π_2EXP denotes the class of problems that can be decided by a coNEXP Turing machine with an NP oracle. The class Θ_2P consists of the problems that can be decided by a P Turing machine with nonadaptive calls to an NP oracle (i.e., calls not depending on previous calls).

Next, the Boolean hierarchy BH is the class of languages that can be expressed as a Boolean combination of languages in NP, i.e., every language in BH can be built from finitely many NP languages using intersection and complementation. An alternative characterization of BH (which is then called QH) is as the languages decidable by a P Turing machine using a constant number of calls to an NP oracle.

We use the notation $\mathcal{C}_1 \wedge \mathcal{C}_2$ to denote the conjunctive Boolean combination of two complexity classes (a language from $\mathcal{C}_1 \wedge \mathcal{C}_2$ is the intersection of a language from $\mathcal{C}_1$ and a language from $\mathcal{C}_2$). Using this notation, we define the last two relevant complexity classes, namely DP which is NP $\wedge$ coNP, and its analogue one step higher in the polynomial hierarchy DP$_2$ which is Σ_2P $\wedge$ Π_2P. For further details on the complexity classes recalled above, we refer the reader to [14, 19].

3 Overview of Results

In this section, we give an overview of our results. We start in Section 3.1 with the Consistent Query Answering (CQA) problem, as it is more natural to explain the different types of complexity here. We then continue in Section 3.2 with the Repair Checking (RC) problem.

3.1 Consistent Query Answering

In this section, we discuss the complexity of consistent query answering. Recall that the CQA problem as defined in Section 1 has three parts of input: the database instance, the set of constraints, and the query. Most of the previous research has focused on a particular variant of this problem, where constraints and query are considered fixed, and the input consists only of the database instance. The complexity of this variant is called the *data complexity*.

Yet, as we have three parts of input, there are seven possible variants to consider (as fixing all parts yields a trivial problem). In this section, we study all of the variants of CQA for all the dependency classes introduced in Section 2 and for boolean conjunctive queries as the query language.

In order to unambiguously identify the variants, we now introduce a notation based on [3, 6]. We will write CQA(X) to refer to the variant of consistent query answering where X is fixed. For example, data complexity is denoted as CQA(C,Q), since in this variant, C and Q are fixed. For the variant where no parts of the input are fixed, we simply write CQA rather than CQA(). Another, orthogonal, restriction considered in the literature is bounding the arity of the schema, which we will also consider in this work.

Overview of results. Table 1 shows the complexity of all CQA variants for each of the IC languages from Figure 1. All results in the table are completeness results, with the exception that we typically do not further classify problems in P and undecidable problems. Only for the CQA variants with FDs we show L-membership in order to match the previously known L-membership result for the data complexity of the RC problem for FDs.

Table 1 Complexity of CQA. All entries denote completeness results (except for those inside P and undecidable problems). Black triangles indicate upper (▼) and lower (▲) bounds shown in this paper. White triangles indicate previously known bounds (concrete references can be found in the paragraph on "Known results").

IC $\mathcal{L}$	CQA	CQA(D)	CQA(C)	CQA(Q)	CQA(C,Q)	CQA(D,C)	CQA(D,Q)
FO	undec	undec	undec	undec	undec	undec	undec
$\vee$-tgd	undec	undec	undec	undec	undec	undec	undec
tgd	undec	undec	undec	undec	undec △	undec ▲	undec △
lav tgd	NP ▽	NP	NP	NP	in P ▽	NP	NP ▲
ID	NP	NP	NP	in P ▼	in P	NP △	in P
UC	Π_2EXP ▼	Π_2EXP	Π_2P ▽	Π_2EXP	Π_2P	NP ▼	Π_2EXP
full $\vee$-tgd	Π_2EXP	Π_2EXP	Π_2P	Π_2EXP	Π_2P △	NP	Π_2EXP ▲
full tgd	EXP ▼	EXP	Π_2P ▲	EXP	coNP △▽	NP △	EXP ▲
denial	Π_2P ▽	BH ▼	Π_2P	Π_2P	coNP ▽	NP	BH
egd	Π_2P	BH	Π_2P	Π_2P ▲	coNP	NP	BH ▲
FD	Π_2P	NP ▼	Π_2P	coNP ▽	coNP	NP	in L ▼
key	Π_2P	NP	Π_2P ▲	coNP	coNP △	NP △	in L

Table 2 Complexity of CQA with bounded arity. Complexity for combinations not shown are as in Table 1.

IC $\mathcal{L}$	CQA	CQA(D)	CQA(Q)	CQA(D,Q)
UC	Π_3P ▼	Π_3P	Π_3P	Π_3P
full $\vee$-tgd	Π_3P	Π_3P	Π_3P	Π_3P ▲
full tgd	Π_2P ▼	Θ_2P ▼	Π_2P ▲	Θ_2P ▲

For showing the results claimed in Table 1, it is not necessary to separately show membership and hardness for each single cell. Figure 1 shows the rich inclusion structure between the constraint languages (e.g., every ID is a lav tgd). Recall that lower bounds propagate from more restricted problems to more general ones and upper bounds propagate from more general problems to more restricted ones. Thus, it suffices to show only certain membership and hardness results.

We use triangles in Table 1 to indicate the upper (▽/▼) and lower (△/▲) bounds that have to be shown in order to obtain all results given in the table. Black triangles indicate upper bounds (▼) and lower bounds (▲) shown in this paper. White triangles indicate upper bounds (▽) and lower bounds (△) given in previous work.

▶ **Theorem 2.** *For all variants of the CQA problem studied here, the complexity is as depicted in Table 1.*

We also consider CQA with bounded arity. In most cases, the complexity remains the same as for the unbounded case. Yet, where the unbounded case has provably exponential complexity, the bounded case yields complexity results inside the polynomial hierarchy. Table 2 depicts those cases.

▶ **Theorem 3.** *For the variants of the CQA problem with bounded arity, the complexity is as depicted in Table 2; in all other cases, the complexity is as given in Table 1.*

Known results. In Section 5, we will give the intuition of our results for CQA (black triangles in Tables 1 and 2). In the remainder of this section, we summarize results given in or implicit in previous work (white triangles in Table 1). We proceed from the first to the last column of Table 1, top to bottom.

We first consider CQA (the first column of Table 1). NP-membership for lav tgds follows from [22, Theorem 4.7] as a by-product of the P-membership proof for the data complexity. Π_2P-membership for denial constraints follows from [8], where Π_2P-completeness for key constraints is stated, and it can be checked that it still holds for denial constraints.

Turning to CQA(C) (the third column of Table 1), we have that Π_2P-membership for UCs is implicit in [21, Lemma 4]. For CQA(Q) (the fourth column of Table 1), coNP-membership for FDs is established by a straightforward algorithm that guesses a counter-example and the fact that model-checking for FDs is in L.

We now proceed to CQA(C,Q) (i.e., data complexity, the fifth column of Table 1). Undecidability for tgds is shown in [22, Theorem 7.2] while P-membership for lav tgds is shown in [22, Theorem 4.7]. Π_2P-completeness for UCs is given in [21, Theorem 6]. A slight modification of the proof shows that hardness already holds for full $\vee$-tgds. We now proceed to full tgds. coNP-hardness for full tgds is given in [22, Theorem 5.5]. coNP-membership for full tgds and denial constraints is given in [21]. The matching coNP-hardness for key dependencies is shown in [10, Theorem 3.3].

We next turn to CQA(D,C) (i.e., query complexity, the penultimate column of Table 1). NP-hardness for IDs, full tgds and key dependencies follows trivially from the NP-hardness of conjunctive query answering. The result for IDs was also implicit in [8]. Finally, we proceed to CQA(D,Q) (the last column of Table 1). Undecidability for tgds follows by a slight modification of [22, Theorem 7.2].

3.2 Repair Checking

In this section, we discuss the complexity of repair checking. Recall that the RC problem as defined in Section 1 has three parts of input: two database instances and a set of constraints. As with consistent query answering, previous research has focused on data complexity, which in this case means that the two database instances are the input, while the constraints are considered as fixed.

We use the same notation as for CQA. That is, we write RC(X) to refer to the variant of repair checking where X is fixed. For example, data complexity is denoted as RC(C). For the variant where no part of the input is fixed, we again write RC rather than RC().

While the CQA problem has three dimensions (data, constraints and query), the RC problem only has two dimensions (data and constraints - as both D and I refer to data). For this reason, it is natural to treat both database instances D and I in the same way, i.e. we either fix both or none of them. In fact, it can be shown that fixing only one of the database instances does not lead to any change of complexity. That is, for the considered constraint languages, the problem variants RC(D) and RC(I) have the same complexity as RC. Thus, for every hardness result of RC, we can actually show two hardness results, namely one for RC(D) and one for RC(I). Note however, that we do not explicitly consider RC(D,C) and RC(I,C). Of course, the membership results for RC(C) implicitly carry over to the RC(D,C) and RC(I,C) cases.

Overview of results. Table 3 shows the complexity of the considered RC variants for each of the IC languages from Figure 1. The notation used is as described in Section 3.1 for Tables 1 and 2. Again all results in the table are completeness results, with the exception that we typically do not further classify problems in P. However, in addition to the cases of FDs, we now also establish an L-membership result for IDs in case of the RC(D,I) problem.

Table 3 Complexity of RC. Black triangles indicate upper (▼) and lower (▲) bounds shown in this paper. White triangles indicate previously known bounds (concrete references can be found in the paragraph on “Known results”).

IC $\mathcal{L}$	RC		RC(D,I)		RC(C)	
FO	PSPACE	▼	PSPACE	△	coNP	▽
$\vee$-tgd	Π_3P	▼	DP$_2$	▼	coNP	
tgd	Π_3P	▲	DP$_2$	▲	coNP	△
lav tgd	DP	▼	DP	▲	in P	▽
ID	in P	▼	in L	▼	in P	
UC	Π_2P	▼	DP	▼	coNP	
full $\vee$-tgd	Π_2P	▲	DP		coNP	△
full tgd	DP	▼	DP	▲	P	△▽
denial	DP	▼	DP		in L	▽
egd	DP		DP	▲	in L	
FD	in L	▼	in L		in L	
key	in L		in L		in L	

▶ **Theorem 4.** *For all variants of the RC problem studied here, the complexity is as depicted in Table 3. The complexity does not change in the case of bounded arity.*

Known results. In Section 4, we will give the intuition of our results for RC (black triangles in Table 3). In the remainder of this section, we summarize results given in or implicit in previous work (white triangles).

First, we consider RC(D,I) (the second column of Table 3). The PSPACE-hardness for FO follows immediately from the PSPACE-hardness of first-order model checking. We now proceed to RC(C) (i.e., data complexity, the last column of Table 3). Staworko [21] showed coNP-completeness for UCs [21, Corollary 3] and P-membership for full tgds [21, Theorem 2]. A slight modification of the proofs shows that coNP-hardness already holds for full $\vee$-tgds. The matching P-hardness for full tgds was given in [2, Theorem 5], where also coNP-hardness for tgds [2, Theorem 7] and L-membership for denial constraints [2, Proposition 5] was proved. The P-membership for lav tgds was given in [22, Theorem 4.9], and the coNP-membership for FO constraints in [2, Proposition 4].

As a concluding remark, note that in Table 3, we do not separately list RC with bounded arity. A quick inspection of our hardness proofs shows that, in case of the Repair Checking problem, the complexity does not change if the arity is bounded.

4 Repair Checking – Intuition

In this section, we will give the intuition and present selected proofs for the repair checking problem. We first illustrate the sources of complexity by discussing the membership results. After that, we will present selected hardness proofs.

The naive algorithm. Repair checking has two fundamental sources of complexity, namely, checking that the supposed repair I is consistent, and checking that it is indeed minimal. This immediately gives the following naive algorithm:

1. check consistency of I
2. check minimality of I by considering the co-problem, where we try to guess a counterexample to minimality (i.e., a consistent instance with smaller symmetric difference)

Since every database instance with a smaller symmetric difference has size at most polynomial in the size of the input, the guess is polynomial. Thus, if we know that the complexity of model checking of a constraint language is in $\mathcal{C}$, then our naive algorithm yields an upper bound of $\mathcal{C} \wedge \mathsf{coNP}^{\mathcal{C}}$ for the variant of RC where all parts of the input vary. Recall that we write $\mathcal{C}_1 \wedge \mathcal{C}_2$ to denote the conjunctive Boolean combination of two complexity classes.

For $\mathcal{C} = \mathsf{NP}$ (which is the case for lav tgds), it it easy to see that the entire second step fits into coNP. Indeed, for the co-problem, one can simultaneously guess a counter-example (a database instance) and a witness for its consistency. In this case, RC is in DP. For other classes $\mathcal{C}$, the $\mathsf{coNP}^{\mathcal{C}}$ factor dominates.

In many cases, one cannot do better than that. In particular, considering the RC problem (i.e., all parts of the input are allowed to vary), we use the upper bound given by the naive algorithm to show PSPACE-membership for FO, Π_3P-membership for $\vee$-tgds, DP-membership for lav tgds, and Π_2P-membership for UCs (four of the ▼ in the first column of Table 3).

Beyond the naive algorithm. In some cases, one *can* do better than the naive algorithm. For full tgds, [21, Lemma 1] provides an NP test for checking minimality of a candidate repair. Since model checking for full tgds is in coNP, we get a DP algorithm for RC.

For denial constraints, the minimality check only needs to test if all immediate supersets are inconsistent. This is the case since denial constraints can only be repaired by deletions and since they are monotone in the sense that supersets of inconsistent instances are always inconsistent. Since consistency can be checked in coNP for denial constraints, we thus have a DP algorithm for RC. For FDs, the tractability of consistency checking yields a polynomial-time (actually, even a logarithmic-space) algorithm for RC.

For IDs, P-membership for RC exploits the existence of a unique subset repair (subset repairs are those repairs that can be obtained by deletions only). Such a subset repair can be computed in polynomial time in case of IDs. Using a construction similar to the one given in [22, Lemma 4.8], we can exploit subset repairs to devise a polynomial-time algorithm for the RC problem of IDs.

Fixing the instances. If we fix the instances D and I, that is, if we consider $RC(D, I)$, the naive algorithm can be refined into:

1. check consistency
2. for every instance I' between D and I, check $I' \not\models C$

Observe that the second step of the algorithm has turned from a guess into an explicit computation. In total, this refined version yields an upper bound of $\mathcal{C} \wedge \mathsf{co}\mathcal{C}$. Let us now consider the results for RC(D,I) (the second column of Table 3): Using this algorithm, we obtain DP_2-membership for $\vee$-tgds and DP-membership for UCs. For IDs, we can further improve the P upper bound to an L upper bound. This completes our discussion of the membership results shown in Table 3.

Sources of complexity. An inspection of our proofs yields an interesting relationship between the roles of consistency and minimality checking, our two orthogonal sources of complexity. For tgds and full $\vee$-tgds, minimality checking dominates the complexity of RC. In particular, hardness holds even if the given instance is promised to be consistent. In contrast, for our DP results for RC, the role of consistency and minimality checking is distributed between the NP and the coNP parts of DP (i.e. if one check requires NP power and the other one requires coNP power). As a consequence, if the given instance is promised to be consistent, the complexity of RC for lav tgds drops to coNP while for full tgds and egds it drops to NP.

Selected hardness proofs. We now present two hardness proofs illustrating typical techniques used to obtain our results. Many of our reductions from QSAT problems share similar machinery. We thus define a set and a formula transformation that we will need in most of these proofs. First, we define a set $\hat{c}$ that encodes the legal value-combinations of literals in a clause of a 3CNF formula (i.e., all combinations except for $c(0,0,0)$). Here we identify the truth value true (resp. false) with 1 (resp. 0):

$$\hat{c} = \{c(0,0,1), c(0,1,0), c(0,1,1), c(1,0,0), c(1,0,1), c(1,1,0), c(1,1,1)\}$$

If ψ is a 3CNF formula of the form $\psi = \bigwedge_i (l_{i1} \vee l_{i2} \vee l_{i3})$, then we denote by ψ^* the conjunction $\bigwedge_i c(l^*_{i1}, l^*_{i2}, l^*_{i3})$ where $l^*_{ij} = x$ if l_{ij} is the positive literal x and $l^*_{ij} = \overline{x}$ if l_{ij} is the negative literal $\neg x$. For example, $[(x \vee \neg z \vee y) \wedge (\neg z \vee y \vee \neg y)]^* = c(x, \overline{z}, y) \wedge c(\overline{z}, y, \overline{y})$.

▶ **Lemma 5.** *There is a database instance D such that RC(D) for tgds is Π_3P-hard. This holds even for bounded arity and if it is known that $I \models C$.*

Proof. We proceed by reduction from $\exists\mathsf{QSAT}_3$ to the co-problem of RC. Let

$$\varphi = \exists x_1 \dots x_k \, \forall y_1 \dots y_l \, \exists z_1 \dots z_m \, \psi$$

be an arbitrary instance of $\exists\mathsf{QSAT}_3$. W.l.o.g., we may assume that ψ is in 3CNF. From this, we construct an instance (D, I, C) of RC, such that φ is true if and only if I is not a repair of D w.r.t. C.

$$D = \hat{c} \cup \{q(0,1), q(1,0)\} \tag{1}$$

$$I = D \cup \{c(0,0,0)\} \cup \bigcup_{1 \le i \le k} \{p_i(0,1), p_i(1,0)\} \tag{2}$$

$$C = \bigcup_{1 \le i \le k} \{q(x,x') \to \exists y y' \, p_i(y,y')\} \tag{3}$$

$$\cup \bigcup_{1 \le i \le k} \{p_i(x,y) \wedge p_i(y,x) \to c(x,x,x)\} \tag{4}$$

$$\cup \bigcup_{1 \le i \le k} \{q(x,y) \wedge c(x,x,x) \wedge c(y,y,y) \to p_i(x,y)\} \tag{5}$$

$$\cup \{ \bigwedge_{1 \le i \le k} p_i(x_i, \overline{x}_i) \wedge \bigwedge_{1 \le i \le l} q(y_i, \overline{y}_i) \to \exists z_1 \overline{z}_1 \dots z_m \overline{z}_m \bigwedge_{1 \le i \le m} q(z_i, \overline{z}_i) \wedge \psi^* \} \tag{6}$$

It is easy to see that I is a superset of D that is consistent with C. So we claim that φ is true if and only if there is a consistent instance I' with $D \subseteq I' \subsetneq I$. The constraints restrict such an instance I' to a specific form:

1. I' does not contain $c(0,0,0)$: otherwise by (5) it would contain all of I.
2. I' contains exactly one of $p_i(1,0)$ and $p_i(0,1)$ for all $i \le k$: by (3) I' contains at least one, and (4) would add $c(0,0,0)$ if more than one were present.

The second property establishes a natural 1-to-1 correspondence between such instances and truth assignments to the x_i variables: instance I' corresponds to truth assignment μ with

$$\mu(x_i) = \begin{cases} T & \text{if } p_i(1,0) \in I' \\ F & \text{if } p_i(0,1) \in I' \end{cases}$$

and vice versa. Finally note that I' satisfies C, and in particular the last constraint (6), if and only if $\forall \vec{y} \exists \vec{z} \psi$ is satisfied by the assignment μ corresponding to I'. ◀

We note that while these constraints are not weakly acyclic (see e.g. [22]), the proof can be easily adapted to turn the constraints into a weakly acyclic set of tgds.

▶ **Lemma 6.** *There are database instances D, I such that RC(D,I) for egds is* DP*-hard. This holds even for bounded arity. If we know that $I \models C$, then this drops to* NP*-hard.*

Proof. We proceed by reduction from 3-colorability and its complement. Let $G = (V, E)$ be an arbitrary instance of 3-colorability and $G' = (V', E')$ be an arbitrary instance of not-3-colorability. W.l.o.g., assume that both E and E' contain the edge $(1, 2)$. From this, we construct the following instance (D, I, C) of RC.

$$\begin{aligned} D &= \{b(1,2), b(1,3), b(2,3), b(2,1), b(3,1), b(3,2), g\} \\ I &= D \setminus \{g\} \\ C &= \{ \bigwedge_{(i,j)\in E} b(x_i, x_j) \land g \to x_1 = x_2, \quad \bigwedge_{(i,j)\in E'} b(x_i, x_j) \to x_1 = x_2\} \end{aligned}$$

Observe how the big conjunctions encode the graphs, and can be satisfied if and only if the corresponding graph is 3-colorable. Since both graphs contain an edge between the vertices 1 and 2, the atom $b(x_1, x_2)$ appears in both conjunctions, ensuring that x_1 and x_2 are assigned different values and thus that the right-hand sides of the egds are false.

Therefore, I is consistent iff G' is not 3-colorable, and D is consistent (thus I not minimal) iff neither G nor G' are 3-colorable. In sum, I is a repair of D w.r.t. C iff G is 3 colorable and G' is not. ◀

5 Consistent Query Answering – Intuition

In this section, we will give the intuition and present selected proofs for the consistent query answering problem. As in the previous section, we first illustrate the sources of complexity by discussing the membership results. After that, we will present selected hardness proofs.

Existential constraints. We first consider existential constraints, i.e., all classes of constraints that allow existential quantification in the conclusion (in Figure 1, these can be found on the left-most branch from FO to ID). For these classes of constraints, we see a particularly clear-cut picture of complexity: They are either undecidable, or have relatively low complexity (in P or NP-complete, depending on the type of complexity considered). The reason for this sharp contrast in complexity is the following: by the monotonicity of CQs, the relevant repairs for CQ-answering are the subset-minimal ones. In case of lav tgds and IDs, we can be sure that all subset-minimal repairs are subsets of the original database instance D. This property is lost for tgds as the following simple example shows:

▶ **Example 7.** Consider the instance (D, C, Q) of CQA with $D = \{a, b\}$ and $C = \{a \to e;\, b \land e \to f\}$ and $Q = b$. In this case, the minimal repairs are $\{b\}$ and $\{a, e\}$. Then we have $\{a, e\} \not\models Q$ and, therefore, $(D, C) \not\models Q$ even though Q is satisfied in all subset-repairs of D w.r.t. C (actually, $\{b\}$ is the only subset-repair). The difficulty comes from the fact that deleting b in a repair only makes sense after e has been added. Such an effect cannot occur with lav tgds. ◀

Consequently, even though lav tgds and IDs also contain existential quantification, all variants of CQA yield complexity of at most NP-completeness. Looking at Table 1, one can see that while known results showed essentially identical pictures for lav tgds and IDs (e.g., CQA is

NP-complete for both, CQA(C,Q) is in P for both), it turns out that for CQA(Q) as well as CQA(D,Q) we have NP-completeness for lav tgds while for IDs we have P-membership. The intuitive reason for P-membership of CQA(Q) with IDs is the existence of a unique subset repair which can be computed in polynomial time for IDs (but the computation requires NP power for lav tgds).

In contrast, for tgds and all extensions thereof (i.e., $\vee$-tgds and FO constraints) undecidability holds for all types of complexity considered here. That is, undecidability holds even if only *one* of the three parts of the input is allowed to vary. Note that there is a close relationship between the CQA problem and the problem of CQ-answering under tgds. In the latter problem, we are given a database D, a set C of tgds and a conjunctive query Q. The question is if D (considered as a conjunction of ground atoms) together with C logically implies Q. It is well-known that the latter problem is undecidable even if (D,Q) or (C,Q) is fixed [15, 7]. From these undecidability results, the undecidability of CQA(D,Q) and of CQA(C,Q) follows immediately. To the best of our knowledge, the undecidability of CQ-answering for fixed (D,C) has not been published so far. It has been observed by G. Gottlob [12] independently of our undecidability proof for CQA(D,C). The key idea of the latter proof is that even for fixed D and C, there can exist arbitrarily big repairs. We can then encode the Halting problem into the CQA(D,C) problem via CQs that ask for the existence of certain chains of binary atoms in every repair.

From universal constraints to full tgds. For universal constraints, the intuition of membership in many cases originates from our algorithm for UCs that we will present next. Previous work [3] showed coN2EXP-membership for CQA (i.e., all parts of the input vary). The algorithm we present here yields a $\mathsf{coNEXP}^{\mathsf{NP}} = \Pi_2\mathsf{EXP}$ upper bound, which together with our hardness results allows us to establish completeness in all cases. Note that [3] considers a semantical definition of UCs that includes logically equivalent FO formulas. Our algorithm also applies to their setting, thus closing the gap left as future work in that paper.

We first illustrate the key ideas of the $\mathsf{coNEXP}^{\mathsf{NP}}$-membership proof for UCs. Let (D, C, Q) be an instance of CQA. The crucial observation is that it never makes sense to introduce fresh domain elements when repairing w.r.t. UCs. More precisely, let G be the set of all ground atoms over the active domain of D. Then every repair of D is a subset of G. We now give the following $\mathsf{NEXP}^{\mathsf{NP}}$ algorithm for the co-problem of CQA.

1. Guess $I \subseteq G$
2. Check that $I \not\models Q$ and $I \models C$
3. Call an oracle to check that there is no J such that $J \models C$ and that J has smaller symmetric difference to D than I

For verifying the complexity of our algorithm, observe that G has at most exponential size. Furthermore, note that checking whether a first-order formula φ is satisfied by a model M can be done in time $O\left(|\varphi|^2 \times |M|^{|\varphi|}\right)$. This model-checking algorithm can also be used inside the NP oracle by padding its input. Thus our algorithm is indeed in $\mathsf{NEXP}^{\mathsf{NP}}$. The cost of the exponential guess in the first step and the call to an NP oracle in the last step remains unchanged even if D and Q are fixed. In contrast, if D and C are fixed, then the complexity drops to NP, i.e., the query complexity of CQ-answering.

Full $\vee$-tgds fall into exactly the same classes of complexity as UCs for all types of complexity – in essence, membership holds for UCs while our hardness results only use full $\vee$-tgds. For full tgds, inspired by [21, Lemma 1], a refined algorithm that exploits the limited number of repairs yields EXP-membership for CQA. Again, the main sources of complexity persist even if D and Q are fixed.

Bounded arity. In Table 2, we observe that only for UCs, full $\vee$-tgds, and full tgds the complexity decreases if we assume bounded arity of all relation symbols involved. More precisely, the complexity drops from the exponential hierarchy (Table 1) to the polynomial hierarchy (Table 2). The reason for this is that if the arity of the relation symbols is bounded by a constant, the number of possible ground atoms over the active domain, and therefore the size of any repair, is polynomially bounded in the size of the input.

The Π_3P upper bound for UCs is obtained by simply revisiting the basic algorithm above and using the fact that now the guess of $I \subseteq G$ in the first step is polynomially bounded. For the repair check (i.e., mainly the minimality check in the third step), a $\Pi_2\mathsf{P}$ oracle is needed. These two sources of complexity persist even for CQA(D,Q).

Also for full tgds, the upper bound is obtained via the basic algorithm. In this case, the repair check drops to DP. We thus get the $\Pi_2\mathsf{P}$ upper bound for CQA with full tgds. In contrast to full $\vee$-tgds, the complexity decreases if we fix the database D. It is convenient to consider a set of full tgds as a datalog program. Then the minimality check only requires the computation of the least fixed point of the immediate consequence operator defined by the datalog program for all (constantly many) subsets of the fixed database D. This fixed point computation can be done with polynomially many nonadaptive oracle calls to an NP oracle. This gives us the $\Theta_2\mathsf{P}$ upper bound in the last line of Table 2.

Less expressive subclasses of UCs. In case of FDs, we have NP-membership for CQA(D), since we can exploit that all repairs are subsets of (a fixed) D. The same holds for the (more expressive) denial constraints, but here we have BH-membership: since model checking is now coNP-hard, a single NP call is not sufficient – but as the subsets are again fixed, no more than constantly many NP oracle calls are needed. The remaining L-membership for CQA(D,Q) with FDs, which we distinguish because known results [2] feature such a more fine-grained analysis, follows from the fact that all repairs are subsets, and repair checking is possible in L.

Selected hardness proofs. The following hardness proof – establishing hardness for any level in the Boolean hierarchy – is of a significantly different flavor than the proofs in Section 4. We will then also present a Π_3P-hardness proof, which uses similar ideas as the proofs in Section 4.

The levels of the Boolean hierarchy BH are denoted by BH_k. In the proof that follows, we will be interested in the co-classes coBH_k that can be defined as follows: $\mathsf{coBH}_1 = \mathsf{coNP}$, $\mathsf{coBH}_2 = \mathsf{coBH}_1 \vee \mathsf{NP}$, $\mathsf{coBH}_3 = \mathsf{coBH}_2 \wedge \mathsf{coNP}$, $\mathsf{coBH}_4 = \mathsf{coBH}_3 \vee \mathsf{NP}$, and so on.

▶ **Lemma 8.** *There is an atomic query Q, such that for every $k > 0$ there is a database instance D s.t. CQA(D,Q) for egds is BH_k-hard. This holds even for bounded arity.*

Proof. We reduce from a coBH_k-hard problem. This suffices since $\mathsf{BH}_k \subseteq \mathsf{coBH}_{k+1}$. As our coBH_k-hard problem, we consider the Boolean combination of 3-colorability. Thus an instance of our problem is given by k graphs $G_1, G_2, \ldots, G_k$ with edges E_i. The question of our problem is whether the Boolean combination of 3-colorability is true. W.l.o.g. let all graphs $G_1, G_2, \ldots, G_k$ contain the edge $(1, 2)$.

We now construct our equivalent CQA(D,Q) instance (D_k, C_k, Q). We first construct Q and D_k. Note that Q is fixed and D_k depends only on k and not on the graphs given in the input of our problem.

$$Q = \exists z_1 z_2 z_3\ a(z_1, z_2, z_3)$$
$$D_k = \{a(1,2,3)\} \cup \bigcup_{1 \leq n \leq k} \{b_n(1,2), b_n(1,3), b_n(2,3), b_n(2,1), b_n(3,1), b_n(3,2)\}$$

Intuitively, the query asks whether there is an a-tuple, and the database D_k consists of an a-tuple and all valid color combinations.

We now proceed to constructing C_k, which will encode the Boolean combination of the k instances of 3-colorability. Recall that the definition of coBH_k alternates between odd ($\mathsf{coBH}_k = \mathsf{coBH}_{k-1} \wedge \mathsf{coNP}$) and even ($\mathsf{coBH}_k = \mathsf{coBH}_{k-1} \vee \mathsf{NP}$) cases. This alternation will be reflected in the construction of C_k. It is convenient to define the following abbreviation: $Col_n = b_n(y_1,y_2) \wedge b_n(y_1,y_3) \wedge b_n(y_2,y_3) \wedge b_n(y_2,y_1) \wedge b_n(y_3,y_1) \wedge b_n(y_3,y_2)$.

$$O_0 = A_0 = \emptyset$$
$$A_n = \begin{cases} A_{n-1} \cup \{a(y_1,y_2,y_3) \wedge \bigwedge_{(i,j)\in E_n} b_n(x_i,x_j)\} & n \text{ odd} \quad (1)\\ \{\varphi \wedge Col_n \mid \varphi \in A_{n-1}\} & n \text{ even} \quad (2) \end{cases}$$
$$O_n = \begin{cases} O_{n-1} & n \text{ odd}\\ O_{n-1} \cup \{\bigwedge_{(i,j)\in E_n} b_n(x_i,x_j)\} & n \text{ even} \quad (3) \end{cases}$$
$$C_k = \{\varphi \rightarrow x_1 = x_2 \mid \varphi \in A_k \cup O_k\}$$

For illustration, here is C_2:

$$\{a(y_1,y_2,y_3) \wedge \bigwedge_{(i,j)\in E_1} b_1(x_i,x_j) \wedge Col_2 \rightarrow x_1 = x_2,\ \bigwedge_{(i,j)\in E_2} b_2(x_i,x_j) \rightarrow x_1 = x_2\}$$

Recall from Lemma 6 how $x_1 = x_2$ in the conclusions can never be fulfilled and thus effectively represent negative constraints. Intuitively, the formula constructed in A_n in the odd case (1) provides a reason to delete $a(1,2,3)$ iff the graph G_n is 3-colorable. The formula constructed in O_n in the even case (3) – together with the additional Col_n conjuncts added in (2) – nullifies any reason to delete $a(1,2,3)$ iff the graph G_n is 3-colorable.

We now proceed to the correctness proof. Let us first note that the query is false in a repair R if and only if there is a formula φ in A_k that is satisfied by $R \cup \{a(1,2,3)\}$. The proof goes by induction and in two cases.

1. k is odd ($\mathsf{coBH}_k = \mathsf{coBH}_{k-1} \wedge \mathsf{coNP}$)
   In this case, there is only one formula added, namely in (1). If G_k is 3-colorable, then this formula is satisfied by D_k and one can avoid deleting one of b_k by deleting $a(1,2,3)$. If on the other hand G_k is not 3-colorable, then the formula cannot be satisfied. So there is a repair of D_k w.r.t. C_k that falsifies the query if and only if there is such a repair of D_{k-1} w.r.t. C_{k-1}. For the base case $k = 1$ note that D_0 is already consistent.
2. k is even ($\mathsf{coBH}_k = \mathsf{coBH}_{k-1} \vee \mathsf{NP}$)
   In this case, there is only one formula added, namely in (3). If G_k is 3-colorable then one of b_k has to be deleted as enforced by the formula. Yet then, by the modifications in (2), no constraint from A_k can ever fire and delete a. On the other hand if G_k is not 3-colorable, then the added formula cannot be satisfied. So there is a repair of D_k w.r.t. C_k that falsifies the query if and only if there is such a repair of D_{k-1} w.r.t. C_{k-1}. ◀

▶ **Lemma 9.** *There is a database instance D and an atomic query Q, s.t. CQA(D,Q) for full $\vee$-tgds is $\Pi_3\mathsf{P}$-hard in case of bounded arity.*

Proof. We proceed by reduction from $\exists$QSAT$_3$ to the co-problem of CQA(D,Q). Let

$$\varphi = \exists x_1 \dots x_k \, \forall y_1 \dots y_l \, \exists z_1 \dots z_m \, \psi$$

be an arbitrary instance of $\exists$QSAT$_3$. W.l.o.g., we may assume that ψ is in 3CNF. From this, we construct the following instance (D, C, Q) of CQA, where $\hat{c}$ and ψ^* are as defined in the paragraph preceding Lemma 5.

$$
\begin{aligned}
D &= \hat{c} \cup \{r(0,1), r(1,0), d(0,1), a\} && (1)\\
C &= \bigcup_{1 \le i \le k} \{d(x,y) \to p_i(x,y) \vee p_i(y,x)\} && (2)\\
&\cup \bigcup_{1 \le i \le l} \{d(x,y) \to q_i(x,y) \vee q_i(y,x)\} && (3)\\
&\cup \bigcup_{1 \le i \le l} \{d(x,y) \wedge b \to q_i(x,y) \wedge q_i(y,x)\} && (4)\\
&\cup \{ \bigwedge_{1 \le i \le k} p_i(x_i, \overline{x_i}) \wedge \bigwedge_{1 \le i \le l} q_i(y_i, \overline{y_i}) \wedge \bigwedge_{1 \le i \le m} r(z_i, \overline{z_i}) \wedge \psi^* \to b\} && (5)\\
&\cup \{d(x,y) \wedge a \wedge b \to e\} && (6)\\
Q &= a && (7)
\end{aligned}
$$

We claim that φ is true iff there is a repair R of D w.r.t. C in which Q is false. Clearly, D is inconsistent since it violates the first $\vee$-tgd in line (2). We distinguish two main cases of repairs, namely either $d(0,1)$ is deleted from D or $d(0,1)$ is retained. If $d(0,1)$ is deleted then there is no reason to delete a. Hence, in these repairs, Q is clearly true. Hence, the only interesting case are repairs which do contain $d(0,1)$.

A repair R containing $d(0,1)$ also contains exactly one of $\{p_i(0,1), p_i(1,0)\}$ for every $i \in \{1, \dots, k\}$. We thus get a 1-to-1 correspondence between the choice of p_i-atoms and truth assignments on $\{x_1, \dots, x_k\}$. Similarly, by the $\vee$-tgd in line (3), at least one of $\{q_i(0,1), q_i(1,0)\}$ for every $i \in \{1, \dots, l\}$ has to be added to R. In case exactly one of $\{q_i(0,1), q_i(1,0)\}$ is added to R, we again get a 1-to-1 correspondence between the choice of q_i-atoms and truth assignments on $\{y_1, \dots, y_l\}$.

Now the crucial question is whether the tgd in line (5) fires and b has to be added to R. If so, then for every $i \in \{1, \dots, l\}$ both $q_i(0,1)$ and $q_i(1,0)$ have to be added to R, due to the tgd in line (4). Note that R thus contains a strict superset of all other choices of q_i-atoms. Due to the minimality of R, the tgd in line (5) thus encodes the following condition: for the chosen p_i-atoms, no matter how we choose one q_i-atom for every $i \in \{1, \dots, l\}$, there exists an instantiation of the variables $(z_i, \overline{z_i})$ to $(0,1)$ or $(1,0)$, such that ψ^* can be matched into the repair R. Finally, note that, since a only occurs in line (6), a repair that deletes a has to contain b.

By making use of the correspondence between p_i and q_i atoms in R on the one hand, and truth assignments to the variables in φ on the other hand, we get the desired equivalence, namely: $(D, C) \not\models Q$ iff there exists a repair R with $a \notin R$ iff there exists a truth assignment μ on $\{x_1, \dots, x_k\}$, s.t. for every extension of μ to $\{y_1, \dots, y_l\}$, there exists a further extension ν to the variables $\{z_1, \dots, z_m\}$, s.t. $\nu \models \psi$, i.e., φ is true. ◀

6 Conclusion

In this work, we have provided a complete picture of the complexity of the RC- and CQA-problems for a wide range of constraint languages. While previous work provided important parts of the picture (in particular, a thorough analysis of data complexity), this work now completes the picture for all types of complexity. In many cases, this has allowed us to get a better understanding of the true sources of complexity.

Tables 1 and 2 summarize the picture for consistent query answering, while Table 3 does the same for repair checking. In particular, for the CQA problem, we get a great variety of complexity results ranging from tractability via various levels of the polynomial hierarchy and various results in the exponential hierarchy up to undecidability. We observe several similarities between classes of constraints such as for UCs and $\vee$-tgds (also for denial constraints and egds). On the other hand, in several cases, we see a diversity of complexity in settings where results for data complexity where relatively uniform. For example, for full tgds, as well as denial constraints and their three subclasses, previously known data complexity results (i.e., CQA(C,Q)) showed coNP-completeness in all five cases (cf. the fifth column of Table 1). Yet if we consider different types of complexity, we see a much more diverse picture, e.g. if we look at CQA(Q) (i.e., the query Q is fixed): full tgds are EXP-complete (Π_2P-complete for bounded arity), while denial constraints and egds are Π_2P-complete and FDs and key dependencies remain coNP-complete (cf. the fourth column of Table 1).

Similar stories could be told about other types of complexity in our tables (for example, in the second column of Table 1 we get BH or Θ_2P instead of Π_2P and we get NP instead of coNP) or for different types of constraint classes considered here. In total, we believe that apart from completing the picture for all types of complexity, the results obtained give new insights into the sources of complexity that were hidden before.

Future work. In this work, we have considered a wide range of constraint languages. However, in the literature, further classes of constraint languages can be found, such as binary constraints [9], weakly acyclic tgds [22], and further subclasses of tgds [17]. The exploration of the combined complexity of the RC- and CQA-problems for ICs from these classes has been left for future work.

Yet more importantly, settings with combinations of various kinds of constraints (such as, e.g., inclusion dependencies with key dependencies) should be further explored, thus extending work that was already started in [8]. Finally, further problem variants deserve future investigation, such as adopting different notions of repairs either by restricting the allowed repair actions or by considering different notions of minimality.

Another natural next question is what happens if not only the arity is bounded, but the whole schema is fixed. In most cases this does not seem to change anything, but some problems (especially regarding UCs and full $\vee$-tgds) indeed become easier. Consider for example RC(D) for UCs: Here, since the schema and the active domain are fixed, all repairs are among a constant set of instances. This allows for an NP minimality check, so the complexity of RC(D) drops to DP - in stark contrast to all cases we considered, where RC(D) and RC always have the same complexity.

Acknowledgments. This work was supported by the Austrian Science Fund (FWF):P25207-N23 and Y698 and by the Vienna Science and Technology Fund (WWTF) project ICT12-15. Sebastian Arming is currently supported by the Austrian Science Fund (FWF): S11411-N23. Emanuel Sallinger is currently supported by the EPSRC grant EP/M025268/1.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: http://www-cse.ucsd.edu/users/vianu/book.html.

2 Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41. ACM Press, 2009. doi:10.1145/1514894.1514899.

3 Marcelo Arenas and Leopoldo Bertossi. On the decidability of consistent query answering. In *AMW*, 2010. URL: http://ceur-ws.org/Vol-619/paper10.pdf.

4 Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79. ACM Press, 1999.

5 Leopoldo Bertossi. Consistent query answering in databases. *ACM SIGMOD Record*, 35(2):68, 2006. doi:10.1145/1147376.1147391.

6 Leopoldo Bertossi. Database Repairing and Consistent Query Answering. *Synthesis Lectures on Data Management*, 2011. doi:10.2200/S00379ED1V01Y201108DTM020.

7 Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *Journal of Artificial Intelligence Research (JAIR)*, 48:115–174, 2013.

8 Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*, pages 260–271. ACM Press, 2003. doi:10.1145/773153.773179.

9 Jan Chomicki. Consistent query answering: Five easy pieces. In *ICDT*, pages 1–17. Springer, 2007. doi:10.1007/11965893_1.

10 Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1-2):90–121, 2005. doi:10.1016/j.ic.2004.04.007.

11 Gaëlle Fontaine. Why is it hard to obtain a dichotomy for consistent query answering? *ACM Trans. Comput. Log.*, 16(1):7:1–7:24, 2015. doi:10.1145/2699912.

12 Georg Gottlob. Personal communication, 2015.

13 Sergio Greco, Fabian Pijcke, and Jef Wijsen. Certain query answering in partially consistent databases. *PVLDB*, 7(5):353–364, 2014. URL: http://www.vldb.org/pvldb/vol7/p353-greco.pdf.

14 David S. Johnson. A catalog of complexity classes. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 67–161. MIT Press, 1990.

15 David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences (JCSS)*, 28(1):167–189, 1984.

16 Paraschos Koutris and Jef Wijsen. The data complexity of consistent query answering for self-join-free conjunctive queries under primary key constraints. In *PODS*, pages 17–29, 2015. doi:10.1145/2745754.2745769.

17 Thomas Lukasiewicz, Maria Vanina Martinez, Andreas Pieris, and Gerardo I. Simari. From classical to consistent query answering under existential rules. In *AAAI*, pages 1546–1552. AAAI Press, 2015.

18 Carsten Lutz and Frank Wolter. On the relationship between consistent query answering and constraint satisfaction problems. In *ICDT*, pages 363–379, 2015. doi:10.4230/LIPIcs.ICDT.2015.363.

19 Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

20 Andreas Pfandler and Emanuel Sallinger. Distance-bounded consistent query answering. In *IJCAI*, pages 2262–2269, 2015. URL: http://ijcai.org/papers15/Abstracts/IJCAI15-320.html.

21 Sławomir Staworko and Jan Chomicki. Consistent query answers in the presence of universal constraints. *Information Systems*, 35(1):1–22, 2010. doi:10.1016/j.is.2009.03.004.

22 Balder ten Cate, Gaëlle Fontaine, and Phokion G. Kolaitis. On the data complexity of consistent query answering. *Theory Comput. Syst.*, 57(4):843–891, 2015. doi:10.1007/s00224-014-9586-0.

23 Jef Wijsen. A survey of the data complexity of consistent query answering under key constraints. In *FoIKS*, pages 62–78, 2014. doi:10.1007/978-3-319-04939-7_2.