Consistent Query Answering in Databases*

Leopoldo Bertossi
Carleton University
School of Computer Science
Ottawa, Canada.
bertossi@scs.carleton.ca

1 Introduction

For several reasons databases may become inconsistent with respect to a given set of integrity constraints (ICs): (a) The DBMS have no mechanism to maintain certain classes of ICs. (b) New constraints are imposed on pre-existing, legacy data. (c) The ICs are soft, user, or informational constraints that are considered at query time, but without being necessarily enforced. (d) Data from different and autonomous sources are being integrated, in particular in mediator-based approaches.

In many cases cleaning the database from inconsistencies may not be an option, e.g. in virtual data integration, or doing it may be costly, non-deterministic, and may lead to loss of potentially useful data. Furthermore, it is quite likely that most of the data in the database is still "consistent". In consequence, an alternative approach to data cleaning consists in basically living with the inconsistent data, but making sure that the consistent data can be identified if desired, for example when queries are answered from the database.

Of course, if this approach is followed, the first problem that has to be confronted is the one characterizing in precise terms the notion of consistent data in a possibly inconsistent database. Such a definition was proposed in [1]. The basic intuition is that a piece of data in the database D is consistent if it is invariant under minimal forms of restoring the consistency of the database, i.e. it remains in every database instance D' that shares the schema with D, is consistent wrt the given ICs, and "minimally differs" from D. This natural intuition still depends on what we consider to be maximally close to the original instance. Several alternatives offer themselves, and we come back to this point in Section 4, but for the moment, in order to fix ideas, we stick to the notion of distance used in [1]: The set of database tuples either inserted or deleted into/from the database to restore consistency has to be made minimal under set inclusion. These repairs will be called S-repairs.

We will concentrate on relational databases. Thus, we start from a fixed relational schema $\mathcal{S} = (\mathcal{U}, \mathcal{R}, \mathcal{B})$, where $\mathcal{U}$ is the possibly infinite database domain, $\mathcal{R}$ is a set of database predicates, and $\mathcal{B}$ is a set of built-in predicates, e.g. comparison predicates. This schema determines a language $\mathcal{L}(\mathcal{S})$ of first-order predicate logic. A database instance D compatible with $\mathcal{S}$ can be seen as a finite collection of ground atoms of the form $R(c_1, ..., c_n)$ that we will call *database tuples*, where R is a predicate in $\mathcal{R}$ and $c_1, ..., c_n$ are constants in $\mathcal{U}$. Built-in predicates have a fixed extension in every database instance, not subject to changes. Integrity constraints are sentences in $\mathcal{L}(\mathcal{S})$.

A database instance D is consistent wrt a finite set IC of integrity constraints if D satisfies IC as a first-order structure, denoted $D \models IC$. Otherwise, D is inconsistent wrt IC. A *repair* D' of D is another instance over schema $\mathcal{S}$ that satisfies IC and makes $\Delta(D, D') = (D \setminus D') \cup (D' \setminus D)$ minimal under set inclusion. We denote with $Rep(D, IC)$ the set of repairs of D wrt IC.

Example 1 Consider a schema with relation $P(A, B, C)$ and the functional dependency (FD) $A \rightarrow B$. The inconsistent instance $D = \{P(a, b, c),\ P(a, c, d),\ P(a, c, e),\ P(b, f, g)\}$ has two repairs: $D_1 = \{P(a, b, c), P(b, f, g)\}$ and $D_2 = \{P(a, c, d),\ P(a, c, e), P(b, f, g)\}$, because the symmetric set differences with the original instance, $\Delta(D, D_1), \Delta(D, D_2)$, are minimal under set inclusion. □

We are not particularly interested in the repairs of the database, or better, in computing them, but in the consistent data in the database, more specifically in consistent answers to queries. From this point of view, we use the repairs as auxiliary objects.[1]

A query is a first-order formula $Q(x_1, \ldots, x_n)$ in $\mathcal{L}(\mathcal{S})$, with free variables $x_1, \ldots, x_n$, $n \geq 0$, and a tuple $\bar{t} = (t_1, \ldots, t_n) \in \mathcal{U}^n$ is a *consistent answer* to Q in D wrt IC if for every $D' \in Rep(D, IC) : D' \models Q[\bar{t}]$, i.e. Q becomes true in D' when the variables $x_1, \ldots, x_n$ take the values $t_1, \ldots, t_n$, resp. When $n = 0$, i.e. the query is boolean, *yes* is the consistent answer if $D' \models Q$ for every $D' \in Rep(D, IC)$. Otherwise, *no* is the consistent answer.

Example 2 (example 1 continued) The query $Q_1(x) : \exists y \exists z P(x, y, x)$ has $(a), (b)$ as the only consistent answers, because they are the only usual answers obtained from the two repairs. The query $Q_2(y, z) : \exists x P(x, y, z)$

* **Database Principles Column.**

Editor: Leonid Libkin, Department of Computer Science, University of Toronto, Toronto, Canada M5S 3H5. E-mail: libkin@cs.toronto.edu.

[1] However, in some applications computing repairs may be more relevant that computing consistent answers to queries, cf. Section 4.1.

has (f, g) as consistent answer. Finally, $Q_3 : P(x, y, x)$ has (b, f, g) as its consistent answer. □

Since these definitions were introduced in [1], several papers have been written on the subject of *consistent query answering* (CQA). For an earlier survey, more references, and related work, see [12].

2 Computing Consistent Answers

As emphasized above, in most applications, computing consistent answers to queries is the natural problem to solve. In an ideal situation, those answers should be obtained by querying the given, inconsistent databases, hopefully avoiding as much as possible the explicit computation of the repairs and checking candidate answers in them. It is easy to see that there may be exponentially many repairs in the size of the database.

The first algorithm for computing consistent answers to queries in this spirit was proposed in [1]. It is a query rewriting algorithm, i.e. the original query Q, expecting for consistent answers from D, is rewritten into a new query Q' in such a way that the usual answers to Q' in D are the consistent answers to Q from D. Query Q' can be computed iteratively by appending to Q additional conditions, the *residues*, that are obtained from the interaction between Q and the ICs, in order to locally enforce the satisfaction of the ICs.

Example 3 (example 2 continued) Query Q_3 can be rewritten into the query
$Q'(x, y, z)$: $P(x, y, z) \wedge \forall v \forall w(\neg P(x, v, w) \vee y = v)$, asking for those tuples (x, y, z) in P for which x does not have and associated u different from y. The *residue* can be obtained by resolution between the original query and the clausal form $\forall uvv'ww'(\neg P(u, v, w) \vee \neg P(u, v', w') \vee v = v')$ of the FD. □

In this example the original query was rewritten into a new first-order query, that could be expressed in SQL, as the original query, and posed to instance D. The usual answers to the rewritten query are exactly the consistent answers to the original query.

Notice that there may be several ICs and also several literals in the query that "logically interact" with the ICs, so that this residue appending process has to be iterated. In [1] conditions for soundness, completeness, and finite termination are identified. This algorithm is guaranteed to produce a first-order rewriting for projection-free conjunctive queries and universal ICs, i.e. that are logically equivalent to a quantifier-free formula preceded by a block of universal quantifiers.

A system implementation for computing consistent answers for this class of queries and ICs that is based on this methodology is reported in [21]. Another system for the same class of queries, but for denial ICs, i.e. ICs that logically equivalent to one of the form

$$\forall \bar{x} \neg (A_1 \wedge \ldots \wedge A_m \wedge \gamma), \qquad (1)$$

where each A_i is a database atom and γ is a conjunction of comparison atoms, is described in [22], however the latter is based on a graph-theoretic representation of repairs and not on syntactic query rewriting (cf. Section 3).

Given the -as we will see- intrinsic limitations of methodology to obtain consistent answers based on first-order query rewriting, a more general approach based on specifying database repairs as the the models of a logic program was proposed [4, 35]. More specifically, it is possible to use a disjunctive logic program with stable model semantics [34] and annotation constants, whose rules capture the violations of the ICs in the bodies and propose alternative ways of locally restoring consistency by means of their disjunctive heads [5].

Example 4 (example 1 continued) The repair program[2] $\Pi(D, FD)$ contains the original database atoms as facts, and the rule $P'(x, y, z, \mathbf{d}) \vee P'(x, u, v, \mathbf{d}) \leftarrow P(x, y, z), P(x, u, v), y \neq u$, where the extra annotation $\mathbf{d}$ used in the head indicates that one of the conflicting tuples should be deleted. The body is satisfied when the FD is violated.

The annotation constant $\mathbf{r}$ is used to read off the database tuples that are inside a repair $P(\bar{x}, \mathbf{r}) \leftarrow P(\bar{x}),\ not\ P'(\bar{x}, \mathbf{d})$, i.e. they contain those database atoms that were not deleted from the original database.

The stable models of the repair program are in one-to-one correspondence with the database repairs; and if the consistent answers to the query $\exists y \exists z P(x, y, z)$ are to be computed, the repair program can be extended with the query rule $Ans(x) \leftarrow P'(x, y, z, \mathbf{r})$. The consistent answers will be those in the extension of the Ans predicate when the extended program is evaluated according to the skeptical (or cautious or certain) stable model semantics, i.e. the one that sanctions as true whatever is true of all stable models. □

This methodology is provably correct and complete for any combination of first-order queries (or in extensions of Datalog), universal ICs, and referential ICs; the latter containing no referential cycles [6]. In spite of its generality, this approach may be too expensive given that query evaluation of disjunctive logic programs under the skeptical stable model semantics is Π_2^P-complete [24]. However, as we will see in Section 3, in some cases we need the expressive power of this kind of programs. Optimizations of these repairs programs have been presented in [27, 6, 20].

3 Complexity of Consistent Query Answering

It is clear that for those cases where the first-order query rewriting methodology for CQA works, the complexity of computing consistent answers is polynomial in the size

[2] In a very simplified form wrt the general methodology, to show the gist of the approach.

$|D|$ of the underlying database D, i.e. it is in $PTIME$ in *data complexity*, a usual measure of complexity in databases.

The first lower-bounds for CQA were obtained for sets FD of functional dependencies and scalar (group-by free) aggregate queries [2]. Since in this case the query returns a single numerical value from a database, which can be different in every repair, the semantics for CQA used in this context is the *range semantics*. In this case the consistent answer to an aggregate query Q is an optimal numerical interval $I_{op} = (op_L, op_U)$, such that for every $D' \in Rep(D, IC)$, the numerical answer $Q(D')$ to Q in D' falls in I_{op}.

Example 5 Consider schema $P(A, B)$, the FD $A \rightarrow B$, and the inconsistent instance $D = \{P(a, 4), P(a, 9), P(b, 3), P(c, 5)\}$. The repairs are $D_1 = \{P(a, 4), P(b, 3), P(c, 5)\}$ and $D_2 = \{P(a, 9), P(b, 3), P(c, 5)\}$. Here, the consistent answer to the query $min(B)$ should be 3, the minimum in both repairs, whereas for the query $sum(B)$, we get answers 13 and 17 from D_1, D_2, resp. Under the range semantics for CQA, the answer is the optimal interval $[13, 17]$, the most informative interval where the answers to the original query from the different repairs will be found. □

Finding the optimal lower and upper bounds op_L, op_U, resp., become maximization and minimization problems, resp., that have associated decision problems: $CQA_L(Q, FD) = \{(D, k) \mid k \leq op_L\}$, and $CQA_U(Q, FD) = \{(D, k) \mid op_U \leq k\}$, resp. As usual, we are interested in the data complexity of these problems.

In order to obtain complexity bounds and algorithms for these decision problems, it is useful to provide a graph-theoretic characterization of repairs, which are in one-to-one correspondence with the (set-theoretically) maximal independent sets of the *conflict graph* $\mathcal{G}(D, FD)$ associated to instance D and FDs FD. The vertices of $\mathcal{G}(D, FD)$ are the database tuples in D, and any two vertices are connected by an edge if they simultaneously participate in the violation of one of the FDs in FD.

A classification of the complexity of CQA for the main aggregate queries is given in [3]. Basically already with two FDs the decision problems associated to CQA become NP-complete.

For first-order queries $Q(\bar{x})$ the decision problem of CQA is $CQA(Q, IC) = \{(D, \bar{t}) \mid \bar{t}$ is a consistent answer to Q from D wrt $IC\}$.

The graph-theoretic methods introduced in [2] were extended to deal with the CQA problem for sets IC of denial constraints, that include FDs [23]. In this case, instead of conflict graphs we find *conflict hyper-graphs*, whose vertices are the database tuples, but now hyper-edges connect all ad only the tuples that simultaneously violate a denial of the form (1). For denial constraints repairs can be obtained via tuple deletions alone, and there is still a one-to-one correspondence between the S-repairs and the maximal independent sets in the hyper-graph.

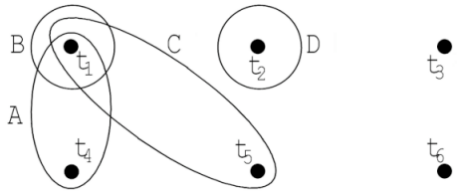
Example 6 The schema has tables $Client(ID, A, C)$, with attributes ID, an identification key, age and credit line; and $Buy(ID, I, P)$, with key $\{ID, I\}$, for clients buying items (I) at certain prices (P). There are denial constraints $IC_1 : \forall ID, P, A, C \neg($ $Buy(ID, I, P)$, $Client(ID, A, C)$, $A < 18$, $P > 25)$ and IC_2: $\forall ID, A, C \neg($ $Client($ $ID, A, C)$, $A < 18, C > 50)$, requiring that people younger than 18 cannot spend more than 25 on one item nor have a credit line higher than 50 in the store. The following is an inconsistent instance D (we use an extra column to denote the tuple)

Client

ID	A	C	
1	15	52	t_1
2	16	51	t_2
3	60	900	t_3

Buy

ID	I	P	
1	CD	27	t_4
1	DVD	26	t_5
3	DVD	40	t_6



The hyper-edges are $\{t_1, t_4\}$ and $\{t_1, t_5\}$ for IC_1 and $\{t_1\}$ and $\{t_2\}$ for IC_2. The figure shows the hyper-graph associated to D and the denial constraints (the key constraints are satisfied in this example). For example, the instance consisting of database tuples $\{t_3, t_4, t_5, t_6\}$ would be an S-repair, that corresponds to maximal independent set. □

In [23] the complexity analysis was extended to denials in combination with inclusion dependencies of the form $\forall \bar{x} \exists \bar{y}(P(\bar{x}) \rightarrow R(\bar{x}', \bar{y}))$, with $\bar{x}' \subseteq \bar{x}$. These constraints can be repaired by deleting the inconsistent tuple in P or inserting a tuple into R. However [23] considers only repairs that are obtained through tuple deletions. This class of dependencies include universal inclusion dependencies and referential ICs. In [23] it shown that for conjunctive queries with projections and FDs, the problem of CQA becomes $coNP$-complete; and with inclusion dependencies, even Π_2^P-complete.

Similar complexity results were obtained in [18] under a repair semantics that allows for both tuple deletions and insertions when FDs and inclusion dependencies are combined. In this case, CQA becomes undecidable due to the possible presence of cycles among the inclusion dependencies and the possibility of using arbitrary elements from the underlying infinite database domain $\mathcal{U}$ when tuples are inserted.

Interestingly, the tight complexity upper bound for decidable CQA coincides with the complexity upper bound

of query evaluation from the repair programs that specify the repairs, which shows that in the worst cases, the expressive power of disjunctive logic programs is necessary.

The complexity results mentioned above show that first-order query rewriting for CQA has intrinsic limitations, which makes it necessary to rewrite the original query into a logic program as opposed to a first-order query. As shown in [6], this program may be simpler than disjunctive. There, classes of ICs are identified for which the disjunctive programs can be transformed into a (non-disjunctive) normal programs, whose query evaluation complexity becomes $coNP$-complete [24].

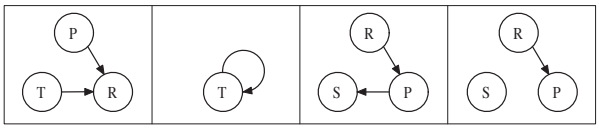
From the previous discussion, we can see that for FDs and conjunctive queries we can go from $PTIME$, e.g. for projection-free conjunctive queries, to $coNP$-completeness for some particular classes of existentially quantified conjunctive queries. In consequence, a natural problem was to identify classes of conjunctive queries with projection for which CQA is still tractable. The first such classes were identified in [23], and polynomial time algorithms were developed appealing to the conflict graph-based representation of repairs. This analysis was extended in detail in [32], were a tight class, $\mathcal{C}_{Tree}$, of conjunctive queries was identified for which CQA is still tractable.

More precisely, as defined in [32], the *join graph* $\mathcal{G}(Q)$ of a boolean conjunctive query Q is a directed graph, whose vertices are the database atoms in Q. There is an arc from L to L' if $L \neq L'$ and there is a variable w that occurs at the position of a non-key attribute in L and also occurs in L'. Furthermore, there is a self-loop at L if there is a variable that occurs at the position of a non-key attribute in L, and at least twice in L. By definition, a query Q belongs to the class $\mathcal{C}_{Tree}$ if Q does not have repeated relations symbols and $\mathcal{G}(Q)$ is a forest and every non-key to key join of Q is full i.e. involves the whole key. CQA becomes tractable for queries in $\mathcal{C}_{Tree}$.[3]

Algorithms for consistently answering queries in the class $\mathcal{C}_{Tree}$ wrt FDs were implemented and reported in [33]. Tractable classes of union of queries in $\mathcal{C}_{Tree}$ are identified in [37].

Example 7 Consider the following boolean queries, where the primary keys of the relations involved are underlined and all the variables are existentially quantified: $Q_1 : P(\underline{x}, y) \wedge R(\underline{y}, w) \wedge T(\underline{u}, \underline{v}, y)$; $Q_2 : T(\underline{x}, \underline{y}, y)$; $Q_3 : R(\underline{x}, y) \wedge P(\underline{y}, z) \wedge S(\underline{z}, \underline{u})$; and $Q_4 : R(\underline{x}, y) \wedge S(\underline{w}, z) \wedge P(\underline{y}, u)$. The four associated join graphs, respectively, are shown below. $\mathcal{G}(Q_1)$ and $\mathcal{G}(Q_2)$ are not forests, therefore their queries are not in $\mathcal{C}_{Tree}$. Even though $\mathcal{G}(Q_3)$ is a forest, since it has a non-key to key join that is not full, it does not belong to $\mathcal{C}_{Tree}$.

[3]Open conjunctive queries can be accommodated in this class by treating the free variables in them as constants in the definition of their join graph.



Q_4 is in $\mathcal{C}_{Tree}$ because $\mathcal{G}(Q_3)$ is a forest and all the non-key to key joins are full. □

4 Other Repair Semantics

We have seen that the notion of consistent answer is defined in terms of the auxiliary notion of repair, which includes in its turn a notion of minimal restoration of consistency. Most of the research in CQA has been concentrated on *repairs* that differ from the original database by a *minimal set of whole (inserted or deleted) tuples under set inclusion*, as were defined above.

Depending on the application domain, or complexity issues, other notions of repair could be considered and studied, in particular, in terms of their impact on CQA.

4.1 Attribute-based repairs

In Example 6, if we consider the repair semantics introduced before, inconsistencies would be solved by deleting complete database tuples from the database. In [48, 31] a different kind of repairs was introduced; they are obtained by changing some attribute values in existing tuples. We call them, attribute-based repairs (simply A-repairs). Except for [48], all the A-repairs considered so far [31, 10, 29] minimize a numerical aggregation function over the attribute-value changes.

More precisely, we can use a numerical function w defined on tuples of the form $(R(\bar{t}), A, newValue)$, where R indicates a tuple in the original instance, A the attribute of R, and $newValue$ is the new value that attribute A gets in $R(\bar{t})$. Then, we minimize an aggregation function g over the values taken by w in the instance, that is the A-repairs are consistent instances that minimize the value of g. Typically [31, 29], $w(R(\bar{t}), A, newValue) = \delta(R(\bar{t}).A, newValue)$, i.e. it gives 1 if there is a change, and 0 otherwise. Next g is the sum, that is, the number of changes is minimized.

Most of the research around this kind of repairs has been motivated by the real need to repair the original database without getting rid of the whole conflicting database tuples. Typical applications are related to census data [31, 10], where census forms have to be *edited* in order to make them consistent wrt to certain denial constraints that forbid certain combinations of data. For example, in household data associated to a census, we may consider that there is a mistake if a child under 5 shows "married" as marital status.

In this kind of statistical applications, a main focus of interest is on obtaining concrete repairs. Consistent query answering based on A-repairs is a natural problem for aggregate numerical queries under the range semantics (cf. Section 3).

In [10], the class of attribute repairs was defined in terms of the functions square difference and sum (w and g above, resp.), i.e. repairs minimize the sum of square differences of numerical values in fixable numerical attributes taking integer values. The basic assumptions are that these numerical values are associated to -non necessarily numerical- values taken by a key, and that the key constraint is satisfied by the original database and its repairs (which makes it possible to compare numerical values associated to values of the key). For example, in a census database, this assumption means that a a household code is unique, but values associated to this code may be incorrect wrt to a separate set of denial constraints, e.g. number of people in the house; or the age of the oldest son, etc.

Numerical distances capture in a better way the numerical nature of some relevant attributes, and also the fact that when we repair (in this case, correct mistakes), the corrections are minimal in numerical difference while satisfying the constraints.

Example 8 (example 6 continued) The A-repairs of D that minimize the square distance are

D':

Client

ID	A	C	
1	15	50	t_1'
2	16	50	t_2'
3	60	900	t_3

Buy

ID	I	P	
1	CD	25	t_4'
1	DVD	25	t_5'
3	DVD	40	t_6

D'':

Client

ID	A	C	
1	18	52	t_1''
2	16	50	t_2''
3	60	900	t_3

Buy

ID	I	P	
1	CD	27	t_4
1	DVD	26	t_5
3	DVD	40	t_6

These repairs have the square distances to the original instance $1^2 + 2^2 + 1^2 + 2^2 = 10$ and $1^2 + 3^2 = 10$, resp.; and they satisfy the denial constraints. □

A relevant decision problem to solve in this context consists in determining if there is a repair at a distance smaller than a given budget to the initial instance. This problem turns out to be NP-hard [10] even for the natural class of denial constraints that are "local", as those in Example 6 (intuitively, inconsistencies can be solved by concentrating on changes that are local to each denial). Even more, for this class of constraints there is no polynomial-time approximation schema (PTAS) for the associated optimization problem. However, it is possible to provide a polynomial-time approximation algorithm that gives an answers within a constant factor of the optimal solution.

Another relevant decision problem is CQA for aggregate queries under the range semantics. This problem is $coNP$-hard for the most common scalar aggregate functions, and even for one database atom denial constraints (like IC_2 in Example 6). For example, for sum no PTAS exists, but, again, a polynomial-time and constant factor approximation algorithm can be given for the optimal bounds of the range semantics [10].

Attribute-based repairs under aggregation constraints have been investigated in [29].

4.2 Cardinality-based repairs

Less attention than CQA based on S-repairs (introduced in Section 1) has received the same problem relative to "cardinality-based repairs" of the original database (simply C-repairs) that minimize *the number* of whole database tuples by which the instances differ.

Example 9 (example 1 continued) Among the S-repairs D_1 and D_2, only D_2 is a cardinality-based repair, because the cardinality $|\Delta(D, D_2)|$ of the symmetric set difference becomes a minimum. □

The complexity of CQA wrt denial constraints under the C-repair semantics has been investigated in [44]. Similarly as with S-repairs, the C-repairs are in a one-to-one correspondence with the *maximum* (in cardinality) independent sets (MISs) in the conflict hyper-graph determined by the original instance and the constraints.

A database tuple is consistently true if it belong to every MIS. In contrast to the S-repair semantics, a database tuple in the original instance may not belong to any MIS. Actually we obtain that CQA for ground atomic queries and denial constraints under the C-repair semantics is $P^{NP(log\ n)}$-complete [44], which contrast with the polynomial time complexity for the same problem wrt the S-semantics [23]. That is, for C-repairs we need polynomial-time algorithm that makes $O(log(n))$ calls to an NP-oracle, where n is the size of the original database.

4.3 CQA under null values

The database repairs considered so far have not explicitly taken into account that a database may have null values and that consistency can also be restored using them, e.g. for referential integrity constraints. There are many different semantics for null values and for databases that represent incomplete information. Some of them consider different kinds of null values, or different but labelled nulls of the same kind, etc. In commercial DBMS we find only one null value. There is something like a semantics for its use in the SQL standard, which is not always implemented in real systems.

In [15] this issue was systematically studied, considering databases with possibly many different occurrences of the same $null$ constant, as found in database practice. Also repairs of referential ICs made use of this constant, as an alternative to the co-existing possibility of deleting the violating tuple. This makes it necessary to provide a precise and uniform semantics for IC satisfaction in the presence of null values that extends and is compatible with the way commercial systems deal with ICs. This semantics was introduced in [15], together with a new repair semantics that privileges the use null values over arbitrary constants in the domain when tuples are inserted to satisfy existential ICs.

Example 10 Consider the ICs $\forall xy(P(x,y) \rightarrow T(x))$, $\forall x(T(x) \rightarrow \exists y P(y,x))$, and the inconsistent database $D = \{P(a,b), P(null,a), T(c)\}$. In this case, we have a referential cycle in the set of ICs. According to the modified repair semantics, the four repairs are:

i	D_i
1	$\{P(a,b), P(null,a), T(c), P(null,c), T(a)\}$
2	$\{P(a,b), P(null,a), T(a)\}$
3	$\{P(null,a), T(c), P(null,c)\}$
4	$\{P(null,a)\}$

We obtain a finite number of repairs (each with finite extension). If we repaired the database by using the non-null constants in the infinite domain with the S-repair semantics of [1], we would obtain an infinite number of repairs and infinitely many of them with infinite extension, as considered in [18]. □

In contrast to the undecidability result for CQA under classic S-repair semantics for referential ICs reported in [18], with the null-value based semantics decidability is reached.

5 Dynamic Aspects of CQA

It is in the context of the C-repair semantics (cf. Section 4.2) that dynamic aspects of CQA have been investigated for the first time. More precisely, in [44] the *incremental complexity* of CQA is considered. In this case, the original database D, of size n, is consistent wrt a set of ICs, but after a sequence $U: u_1, \ldots, u_m$ of basic updates u_i that are insertions on one tuple, deletion of one tuple, or a change of an attribute value in a tuple, the updated database $U(D)$ may become inconsistent. Incremental complexity of CQA is the complexity of CQA wrt the updated instance $U(D)$.

For the C-repair semantics, first-order boolean queries, and denial constraints, incremental CQA is in *PTIME* in the size of D, however a naive algorithm that gives this upper bound, namely $O(n^{p(m)})$ with p a polynomial, gives an exponential complexity in m, the size of the update sequence [44].

It is possible to prove that this problem, still under the C-repair semantics, for functional dependencies and ground atomic queries, is *fixed parameter tractable* (FPT) [38, 30, 25], i.e. there is an algorithm that of order $O(f(m) \cdot n^c)$, with f a function of m alone, and c a constant. The parameter here is m, the size of the update sequence. The algorithm given in [44] appeals to the fixed parameter tractability of vertex cover. Using the membership to FPT of the d-hitting set problem (finding the size of a minimum hitting set for a hyper-graph with hyper-edges bounded in size by d) [46], it is possible to prove that incremental CQA under the C-repair semantics also belongs to FPT for denial constraints that have a fixed number d of database tuples, which is a natural assumption [44].

Summarizing, for boolean queries and denial ICs, incremental CQA is $PTIME$ on n under the C-repair semantics. However, under the same conditions but now the S-repair semantics, incremental CQA is $coNP$-complete on n [44]. The difference with the C-repair semantics becomes more clear if we consider that the cost of a C-repair cannot exceed the size m of the update, whereas the cost of an S-repair may be unbounded wrt the same size.

Example 11 Consider the schema $R(\cdot), S(\cdot)$, with denial constraint $\forall x \forall y \neg(R(x) \wedge S(y))$. The following is a consistent instance $D = \{R(1), \ldots, R(n)\}$, with empty table for S. After the update $U: insert(S(0))$, the database becomes inconsistent, and the S-repairs are $\{R(1), \ldots, R(n)\}$ and $\{S(0)\}$. Only the former is a C-repair, at a distance 1 from D, but the second S-repair is at a distance n. □

Finally, we should mention that incremental CQA wrt denial constraints and atomic queries under the A-repair semantics (cf. Section 4.1) becomes P^{NP}-hard on n [44]. In this case, after the update sequence (consisting of insertions, deletions and one attribute value changes), the database may become inconsistent, but it can only be repaired through attribute value changes.

It would be interesting to analyze the complexity of incremental CQA from the point of view of *dynamic complexity* [40, 47] and *incremental complexity* as introduced in [45]. Here the database is not necessarily consistent before the update, but auxiliary data structures can be used for incremental computation of CQA.

6 Data Exchange and Integration

Consistent query answering is related to several other areas, most prominently to virtual data integration, data exchange, and peer-to-peer data exchange.

As emphasized in [11], in data exchange, where data is shipped from a source database in order to populate a target schema, integrity constraints imposed at the target level have to be kept satisfied [41]. Instead of restoring the consistency of data at the target *a posteriori*, after the population process, a more appealing alternative would take the ICs at the target into account when the data mappings between the source and the target are being established and/or used.

In virtual data integration [43] there is no centralized consistency maintenance mechanism that makes the data in the mediated system satisfy certain global ICs. Again, these ICs have to be captured by the mappings between the sources and the global schema or at query time [8], when global queries are being answered. Both in data exchange and virtual data integration, the plans for data transfer or query answering have to deal with potential inconsistencies of data.

Mediator-based data integration is a natural scenario for applying CQA: We expect to retrieve only the information from the global, virtual database that is consistent wrt to a given set of global ICs. In this scenario, some new issues appear, like characterizing repairs of global virtual instances, and retrieve consistent answers at query time, considering that the data is in the sources.

In the following we will illustrate the approach developed in [7, 13, 14] by means of an extended example. For a general overview, detailed discussion of related work, and additional results cf. [8].

We will assume that the *local-as-view* (LAV)[4] approach is adopted, which means that the relations at the source schemas are described as views over the global schema that the mediator offers to the users. The underlying domain is $\mathcal{U} = \{a, b, c, \dots\}$, the global schema is Global system $\mathcal{G}$ consists of relations $P(A, B)$ and $R(C, D)$, and the source relations are S_1, S_2, with extensions $s_1 = \{S(a, b)\}, s_2 = \{S_2(a, c), S_2(d, e)\}$. The source relations are defined by (using Datalog notation):

$$\begin{aligned} S_1(X, Z) &\leftarrow P(X, Y),\ R(Y, Z). \\ S_2(X, Y) &\leftarrow P(X, Y). \end{aligned}$$

The global relations P, R can be materialized in different ways, still satisfying the view definitions (or source descriptions); so different global instances are possible become possible. If the sources are declared as *open* (or incomplete), a global (material) instance D is *legal* if the view definitions applied to it compute extensions $S_1(D), S_2(D)$ such that $s_1 \subseteq S_1(D)$ and $s_2 \subseteq S_2(D)$. That is, each source relation contains a possibly proper subset of the data of its kind in the global system.

The *certain answers* to a global query Q, i.e. expressed in terms of relations P and R, are those that can be (classically) obtained from every possible legal instance. Among the legal instances we can distinguish the *minimal instances*, those that are legal and do not properly contain any other legal instance [7]. Those global instances do not contain redundant data, and we can concentrate on inconsistency already at their level. In our example, the minimal instances are $Mininst(\mathcal{G}) = \{\{P(a, c), P(d, e), P(a, z), R(z, b)\} \mid z \in \mathcal{U}\}$.

The *minimal answers* to a global query Q are those answers that can be obtained from every minimal instance. In consequence, the certain answers to a query are contained in its minimal answers: $Certain(Q, \mathcal{G}) \subseteq Minimal(Q, \mathcal{G})$. For monotone queries these two sets coincide [7].

Now, let us assume that we have the global FD on P: $A \rightarrow B$. Some of the minimal instances are consistent, e.g. $D_1 = \{P(a, c), P(d, e), R(c, b)\}$, but others are not, e.g. $D_2 = \{P(a, c), P(d, e), P(a, f), R(f, b)\}$. The repairs of system $\mathcal{G}$ are defined as the S-repairs of the minimal instances. For example, D_1 here is its own repair, but D_2 gives rise to two repairs: $\{P(a, c), P(d, e), R(f, b)\}$ and $\{P(d, e), P(a, f), R(f, b)\}$. The consistent answers to a global query Q are those answers that can be obtained from every repair of $\mathcal{G}$ [7].

In order to compute consistent answers from $\mathcal{G}$, we proceed as follows [13, 14]: (1) First the minimal instances of $\mathcal{G}$ are specified as the stable models of a logic program. (2) The repairs of $\mathcal{G}$ are specified as illustrated in Section 2. This constitutes a second layer on top of the program in (1). (3) A third layer is the query program that is run in combination with the other two previous layer. This program contains an $Answer$ predicate that collects the consistent answers, that are computed according to the skeptical stable model semantics of the combined program.

This methodology provably works for first-order queries (and Datalog extensions), and universal ICs combined with (acyclic) referential ICs. As an interesting subproduct we obtain a methodology to compute certain answers to monotone queries (just forget the intermediate repair layer).

In our example, the minimal instances are specified by the program containing the following rules (in addition to the facts $dom(a)$, $dom(b)$, $dom(c)$, $\dots$, $S_1(a, b)$, $S_1(d, e)$, $S_2(a, c)$):

$$\begin{aligned} P(X, Z) &\leftarrow S_1(X, Y), F((X, Y), Z). \\ P(X, Y) &\leftarrow S_2(X, Y). \\ R(Z, Y) &\leftarrow S_1(X, Y), F((X, Y), Z). \\ F((X, Y), Z) &\leftarrow S_1(X, Y), dom(Z), \\ &\qquad choice((X, Y), (Z)). \end{aligned}$$

This program is inspired by the inverse rules algorithm for computing certain answers [26]. Here, F is a functional predicate, whose functionality on the second argument is imposed through the use of the *choice operator* $choice((\bar{X}), (Z))$, that non-deterministically chooses a unique value for Z for each combination of values for $\bar{X}$ [36]. The program with choice can be transformed into a usual normal program with stable model semantics [36]. In our case, there is a one-to-one correspondence between its models and the minimal instances of the integration system [14]. This program not only specifies the minimal instances, but, in combination with a query program, can be also used to compute certain answers to monotone queries. Specification programs can also be produced for case where sources may be *closed* (or *complete*) or *clopen* (or exact) [8].

Now, in order to specify the repairs of $\mathcal{G}$, we can use a program like the one presented in Example 4, as a second layer on top of the previous program.

In a series of papers [17, 16, 42] virtual data integration systems under the *global-as-view* (GAV) approach (global relations are defined as views over the source relations). Different alternative semantics are studied and different classes of global ICs are considered. For example, in their

[4]For the basic notions of virtual data integration we refer to [43].

terminology, the repairs of D_2 would be *loosely sound*, in the sense that, as global instances, they are not legal because the computed views do not extend the given source contents.

In semantic approaches to peer-to-peer data exchange systems [39], peers exchange data according to declarative mappings between them, with the purpose of complementing a peer's data when a local queries posed to peers have to be answered. The exchange of data is governed by theses data mappings, mappings derived by transitivity, and a peer's own local constraints that have to be respected when data from other peers is received [9, 19]. Even more, there may be certain *trust relationships* between peers that should also be taken into account. When inconsistencies are "solved", a peer may decide to trust one peer more than the others that are contributing with data [9].

Acknowledgments: Research supported by NSERC, and CITO/IBM-CAS. L. Bertossi is Faculty Fellow of IBM Center for Advanced Studies (Toronto Lab.). Several contributions and comments by Loreto Bravo are very much appreciated.

References

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

[2] Arenas, M., Bertossi, L. and Chomicki, J. Scalar Aggregation in FD-Inconsistent Databases. *Proc. International Conference on Database Theory (ICDT 01)*, Springer LNCS 1973, 2001, pp. 39-53.

[3] 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.

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

[5] Barcelo, P. and Bertossi, L. Logic Programs for Querying Inconsistent Databases. *Proc. Practical Aspects of Declarative Languages (PADL 03)*, Springer LNCS 2562, 2003, pp. 208-222.

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

[7] Bertossi, L., Chomicki, J., Cortes, A. and Gutierrez, C. Consistent Answers from Integrated Data Sources. *Proc. Flexible Query Answering Systems (FQAS 02)*, Springer LNCS 2522, 2002, pp. 71–85.

[8] Bertossi, L. and Bravo, L. Consistent Query Answers in Virtual Data Integration Systems. In *Inconsistency Tolerance*, Springer LNCS 3300, 2004, pp. 42-83.

[9] Bertossi, L. and Bravo, L. Query Answering in Peer-to-Peer Data Exchange Systems. *Proc. EDBT International Workshop on Peer-to-Peer Computing & DataBases (P2P&DB 04)*, Springer LNCS 3268, 2004, pp. 478-485.

[10] Bertossi, L., Bravo, L., Franconi, E. and Lopatenko, A. Fixing Numerical Attributes under Integrity Constraints. *Proc. International Symposium on Database Programming Languages (DBPL 05)*, Springer LNCS 3774, 2005, pp. 262-278.

[11] Bertossi, L., Chomicki, J., Godfrey, P., Kolaitis, Ph., Thomo, A. and Zuzarte, C. Exchange, Integration, and Consistency of Data: Report on the ARISE/NISR Workshop. *SIGMOD Record*, 2005, 34(3):87-90.

[12] Bertossi, L. and Chomicki, J. Query Answering in Inconsistent Databases. In *Logics for Emerging Applications of Databases*. Springer, 2003, pp. 43-83.

[13] Bravo, L. and Bertossi, L. Logic Programs for Consistently Querying Data Integration Systems. *Proc. International Joint Conference in Artificial Intelligence (IJCAI 03)*, Morgan Kauffmann Publishers, 2003, pp. 10-15.

[14] Bravo, L. and Bertossi, L. Disjunctive Deductive Databases for Computing Certain and Consistent Answers to Queries from Mediated Data Integration Systems. *Journal of Applied Logic*, 2005, 3(2):329-367.

[15] 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)*. To appear in Springer LNCS. Technical Report arXiv:cs.DB/0604076 v1. Posted April 19, 2006.

[16] Cali, A., Calvanese, D., De Giacomo, G. and Lenzerini, M. Data Integration Under Integrity Constraints. In *Proc. Conference on Advanced Information Systems Engineering (CAISE 02)*, Springer LNCS 2348, 2002, pp. 262–279.

[17] Cali, A., Calvanese, D., De Giacomo, G. and Lenzerini, M. On the Role of Integrity Constraints in Data Integration. *IEEE Data Engineering Bulletin*, 2002, 25(3): 39-45.

[18] Calì, 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)*, ACM Press, 2003, pp. 260–271.

[19] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M. and Rosati, R. Inconsistency Tolerance in P2P Data Integration: An Epistemic Logic Approach. *Proc. International Symposium on Database Programming Languages (DBPL 05)*, Springer LNCS 3774, 2005, pp. 90-105.

[20] Caniupan, M. and Bertossi, L. Optimizing Repair Programs for Consistent Query Answering. *Proc. International Conference of the Chilean Computer Science Society (SCCC 05)*, IEEE Computer Society Press, 2005, pp. 3-12.

[21] Celle, A. and Bertossi, L. Querying Inconsistent Databases: Algorithms and Implementation. In *Computational Logic - CL 2000*, Springer LNCS 1861, 2000, pp. 942-956.

[22] Chomicki, J., Marcinkowski, J. and Staworko, S. Computing Consistent Query Answers using Conflict Hypergraphs. *Proc. Conference on Information and Knowledge Management (CIKM 04)*, ACM Press, 2004, pp. 417 - 426.

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

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

[25] Downey, R.G. and Fellows, M.R. *Parameterized Complexity*. Springer, Monographs in Computer Science, 1999.

[26] Duschka, O., Genesereth, M. and Levy, A. Recursive Query Plans for Data Integration. *Journal of Logic Programming*, 2000, 43(1):49-73.

[27] Eiter, T., Fink, M., Greco, G. and Lembo, D. Efficient Evaluation of Logic Programs for Querying Data Integration Systems. *Proc. International Conference on Logic Programming (ICLP 03)*, Springer LNCS 2916, 2003, pp. 163-177. 2916, 2003.

[28] Fagin, R., Kolaitis, Ph. G., Miller, R. J., and Popa, L. Data Exchange: Semantics and Query Answering. *Proc. International Conference on Database Theory (ICDT)*. 207–224, 2003.

[29] Flesca, S., Furfaro, F. Parisi, F. Consistent Query Answers on Numerical Databases under Aggregate Constraints. *Proc. International Symposium on Database Programming Languages (DBPL 05)*, Springer LNCS 3774, 2005, pp. 279-294.

[30] Flum, J. and Grohe, M. Parameterized Complexity Theory. Springer, Texts in Theoretical Computer Science, 2006.

[31] Franconi, E., Laureti Palma, A., Leone, N., Perri, S. and Scarcello, F. Census Data Repair: a Challenging Application of Disjunctive Logic Programming. *Proc. Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 01)*. Springer LNCS 2250, 2001, pp. 561-578.

[32] Fuxman, A. and Miller, R. First-Order Query Rewriting for Inconsistent Databases. *Proc. International Conference on Database Theory (ICDT 05)*, Springer LNCS 3363, 2004, pp. 337-351.

[33] Fuxman, A., Fazli, E. and Miller, R.J. ConQuer: Efficient Management of Inconsistent Databases. *Proc. ACM International Conference on Management of Data (SIGMOD 05)*, ACM Press, 2005, pp. 155-166.

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

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

[36] Giannotti, F., Greco, S., Sacca, D. and Zaniolo, C. Programming with Non-determinism in Deductive Databases. *Annals of Mathematics and Artificial Intelligence*, 1997, 19(1-2):97-125.

[37] Grieco, L., Lembo, D., Rosati, R. and Ruzzi, M. Consistent Query Answering under Key and Exclusion Dependencies: Algorithms and Experiments. *Proc. ACM International conference on Information and Knowledge Management (CIKM 05)*, ACM Press, 2005, pp. 792-799.

[38] Grohe, M. Parameterized Complexity for the Data-base Theorist. *SIGMOD Record*, 2002, 31(4):86-96.

[39] Halevy, A., Ives, Z., Suciu, D. and Tatarinov, I. Schema Mediation in Peer Data Management Systems. *Proc. Intenational Conference on Data Engineering (ICDE 03)*, IEEE Computer Society Press, 2003, pp. 505-518.

[40] Immerman, N. *Descriptive Complexity*. Graduate Texts in Computer Science. Springer, 1999.

[41] Kolaitis, Ph. Schema Mappings, Data Exchange, and Metadata Management. *Proc. ACM Symposium on Principles of Database Systems (PODS 05)*, ACM Press, 2005, pp. 61-75.

[42] Lembo, D., Lenzerini, M. and Rosati, R. Source Inconsistency and Incompleteness in Data Integration. In *Proc. International Workshop Knowledge Representation meets Databases (KRDB 02)*, CEUR Electronic Workshop Proceedings, 2002.

[43] Lenzerini, M. Data Integration: A Theoretical Perspective. *Proc. ACM Symposium on Principles of Database Systems (PODS 02)*, ACM Press, 2002, pp. 233-246

[44] Lopatenko, A. and Bertossi, L. Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics. Technical Report arXiv:cs.DB/0604002 v1. Posted April 2, 2006.

[45] Miltersen, P.B., Subramanian, S., Vitter, J.S. and Tamassia, R. Complexity Models for Incremental Computation. *Theoretical Computer Science*, 1994, 130(1):203-236.

[46] Niedermeier, R. and Rossmanith, P. An Efficient Fixed-Parameter Algorithm for 3-Hitting Set. *Journal of Discrete Algorithms*, 2003, 1(1):89-102.

[47] Weber, V. and Schwentick, T. Dynamic Complexity Theory Revisited. *Proc. Annual Symposium on Theoretical Aspects of Computer Science (STACS 05)*, Springer LNCS 3404, 2005, pp. 256-268.

[48] Wijsen, J. Condensed Representation of Database Repairs for Consistent Query Answering. *Proc. International Conference on Database Theory (ICDT)*, pages 378–393. Springer-Verlag, LNCS 2572, 2003.