

On the Computational Complexity of Minimal-Change Integrity Maintenance in Relational Databases*

Jan Chomicki¹ and Jerzy Marcinkowski²

¹ Dept. of Computer Science and Engineering
University at Buffalo
Buffalo, NY 14260-2000

`chomicki@cse.buffalo.edu`

² Instytut Informatyki

Wrocław University

51-151 Wrocław, Poland

`Jerzy.Marcinkowski@ii.uni.wroc.pl`

Abstract. We address the problem of minimal-change integrity maintenance in the context of integrity constraints in relational databases. Using the framework proposed by Arenas, Bertossi, and Chomicki [4], we focus on two basic computational issues: *repair checking* (is a database instance a repair of a given database?) and *consistent query answers* (is a tuple an answer to a given query in every repair of a given database?). We study the computational complexity of both problems, delineating the boundary between the tractable and the intractable. We review relevant semantical issues and survey different computational mechanisms proposed in this context. Our analysis sheds light on the computational feasibility of minimal-change integrity maintenance. The tractable cases should lead to practical implementations. The intractability results highlight the inherent limitations of any integrity enforcement mechanism, e.g., triggers or referential constraint actions, as a way of performing minimal-change integrity maintenance.

1 Introduction

Inconsistency is a common phenomenon in the database world today. Even though integrity constraints successfully capture data semantics, the actual data in the database often fails to satisfy such constraints. This may happen because the data is drawn from a variety of independent sources as in data integration (see Lenzerini's survey [57]), or the data is involved in complex, long-running activities like workflows.

How to deal with inconsistent data? The traditional way is to not allow the database to become inconsistent by aborting updates or transactions leading to

* This material is based upon work supported by the National Science Foundation under Grant No. IIS-0119186 and UB start-up funds.

integrity violations. We argue that in present-day applications this scenario is becoming increasingly impractical. First, if a violation occurs because of data from multiple, independent sources being merged (a scenario identified by Lin and Mendelzon [58]), there is no single update responsible for the violation. Moreover, the updates have typically already committed. For example, if we know that a person should have a single address but multiple data sources contain different addresses for the same person, it is not clear how to fix this violation through aborting some update. Second, the data may have become inconsistent through the execution of some complex activity and it is no longer possible to trace the inconsistency to a specific action.

In the context of triggers or referential integrity, more sophisticated methods for handling integrity violations have been developed. For example, instead of being aborted an update may be propagated. In general, the result is at best a consistent database state, typically with no guarantees on its distance from the original, inconsistent state (the research of Ludäscher, May, and Lausen [59] is an exception).

In our opinion, integrity restoration should be a separate process that is executed after an inconsistency is detected. Such an approach is also advocated by Embury et al. [36] under the name of *data reconciliation*. The restoration should have a minimal impact on the database by trying to preserve as many tuples as possible. This scenario is called from now on *minimal-change integrity maintenance*.

We claim that a central notion in the context of integrity restoration is that of a *repair* [4]. A repair is a database instance that satisfies integrity constraints and minimally differs from the original database (which may be inconsistent).

One can interpret the postulate of minimal change in several different ways, depending on whether the information in the database is assumed to be *correct* and *complete*. If the information is complete but not necessarily correct (it may violate integrity constraints), the only way to fix the database is by *deleting* some parts of it. This is a common approach in data warehousing. On the other hand, if the information is both incorrect and incomplete, then both insertions and deletions should be considered. Thus, in some data integration approaches, for example the work of Lenzerini, Lembo, and Rosati [56,57], the completeness assumption is not made. For large classes of constraints, e.g., denial constraints, the restriction to deletions has no impact, since only deletions can remove integrity violations. Another dimension of change minimality is whether updates to selected attributes of tuples are considered as a way to remove integrity violations.

Regardless of what notion of minimal change is assumed, a basic computational problem in the context of integrity maintenance is *repair checking*, namely checking whether a given database instance is a repair of the original database. The complexity of this problem, under different notions of minimal change, is studied in the present paper. Repair checking algorithms can typically be converted to algorithms for nondeterministically computing repairs.

Sometimes when the data is retrieved online from multiple, autonomous sources, it is not possible to restore the consistency of the database by constructing a single repair. In that case one has to settle for computing, in response to queries, *consistent query answers* [4], namely answers that are true in every repair of the given database. Such answers constitute a conservative “lower bound” on the information present in the database. The problem of computing consistent query answers is the second computational problem studied in the present paper. The notion of consistent query answer proposed by Arenas, Bertossi and Chomicki [4] has been used and extended in many papers [5,6,7,8,9,11,12,16,19,20,21,25,26,27,34,41,45,46,49,62,64]. This research has been surveyed by Bertossi and Chomicki [13].

We describe now the setting of our results. We analyze the computational complexity of repair checking and consistent query answers along several different dimensions. We characterize the impact of the *class of queries* and the *class of integrity constraints* under consideration.

Our results shed light on the computational feasibility of minimal-change integrity maintenance. The tractable cases should lead (and to some degree already have led) to practical implementations. The intractability results highlight the inherent limitations of any integrity enforcement mechanism, e.g., triggers or referential constraint actions [59,60], as ways of performing minimal-change integrity maintenance.

The plan of the paper is as follows. In the first three sections, we discuss first-order (or equivalently: relational algebra) queries. In Section 2, we define the basic framework. In Section 3, we consider denial constraints, for which repairs are obtained by deleting facts. In Section 4, we discuss more general universal constraints and inclusion dependencies, under different notions of repair. In Section 5, we study aggregation queries in the presence of functional dependencies. In Section 6, we summarize related research and in Section 7 we draw conclusions and discuss future work. Several key proofs are presented in detail. Other proofs can be found in the original sources.

2 Basic Notions

In the following we assume we have a fixed relational database schema R consisting of a finite set of relations (which are finite sets of tuples). We also have an infinite set of attributes (column labels) U from which relation attributes are drawn. We have a fixed, infinite database domain D , consisting of uninterpreted constants, and an infinite numeric domain N consisting of all rational numbers. Those domains are disjoint. The database instances can be seen as finite, first-order structures over the given schema, that share the domains D and N . Every attribute in U is typed, thus all the instances of R can only contain in a single attribute either uninterpreted constants or numbers. Since each instance is finite, it has a finite active domain which is a subset of $D \cup N$. (The property that attribute values are atomic is often called *First Normal Form* or *1NF*.) As usual, we allow the standard built-in predicates over N ($=, \neq, <, >, \leq, \geq$) that

have infinite, fixed extensions. With all these elements we can build a first order language \mathcal{L} .

2.1 Integrity Constraints

Integrity constraints are closed first-order \mathcal{L} -formulas. In the sequel we will denote relation symbols by P, P_1, \dots, P_m , tuples of variables and constants by $\bar{x}_1, \dots, \bar{x}_m$, and quantifier-free formulas referring to built-in predicates by φ .

In this paper we consider the following basic classes of integrity constraints:

1. *Universal integrity constraints*: \mathcal{L} -sentences

$$\forall \bar{x}_1, \dots, \bar{x}_n. \left(\bigvee_{i=1}^m P_i(\bar{x}_i) \vee \bigvee_{i=m+1}^n \neg P_i(\bar{x}_i) \vee \varphi(\bar{x}_1, \dots, \bar{x}_n) \right).$$

We assume that all variables appearing in positive literals appear also in at least one negative literal.

2. *Denial constraints*: \mathcal{L} -sentences

$$\forall \bar{x}_1, \dots, \bar{x}_n. \left(\bigvee_{i=1}^n \neg P_i(\bar{x}_i) \vee \varphi(\bar{x}_1, \dots, \bar{x}_n) \right).$$

They are a special case of universal constraints.

3. *Binary constraints*: universal constraints with at most two occurrences of database relations.
4. *Functional dependencies (FDs)*: \mathcal{L} -sentences

$$\forall \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \bar{x}_5. \left(\neg P(\bar{x}_1, \bar{x}_2, \bar{x}_4) \vee \neg P(\bar{x}_1, \bar{x}_3, \bar{x}_5) \vee \bar{x}_2 = \bar{x}_3 \right).$$

They are a special case of binary denial constraints. A more familiar formulation of the above FD is $X \rightarrow Y$ where X is the set of attributes of P corresponding to \bar{x}_1 and Y the set of attributes of P corresponding to \bar{x}_2 (and \bar{x}_3).

5. *Referential integrity constraints*, also known as *inclusion dependencies (INDs)*: \mathcal{L} -sentences

$$\forall \bar{x}_1 \exists \bar{x}_3. \left(\neg P_1(\bar{x}_1) \vee P_2(\bar{x}_2, \bar{x}_3) \right),$$

where the \bar{x}_i 's are sequences of distinct variables, with \bar{x}_2 contained in \bar{x}_1 ; and database relations P_1, P_2 . Again, this is often written as $P_1[Y] \subseteq P_2[X]$ where X (resp. Y) is the set of attributes of P_2 (resp. P_1) corresponding to \bar{x}_2 . If P_1 and P_2 are clear from the context, we omit them and write the dependency simply as $Y \subseteq X$. If an IND can be written without any existential quantifiers, then it is called *full*.

Several examples of integrity constraints are presented later in Examples 1, 2, and 8.

Given a set of FDs and INDs IC and a relation P_1 with attributes U_1 , a *key* of P_1 is a minimal set of attributes X of P_1 such that IC entails the FD

$X \rightarrow U_1$. In that case, we say that each FD $X \rightarrow Y \in IC$ is a *key dependency* and each IND $P_2[Y] \subseteq P_1[X] \in IC$ (where P_2 is also a relation) is a *foreign key constraint*. If, additionally, X is the primary (one designated) key of P_1 , then both kinds of dependencies are termed *primary*.

The above constraint classes are the most common in database practice. They exhaust the constraints supported by present-day database management systems. The SQL:1999 standard [60] proposes general assertions that can be expressed using arbitrary SQL queries (and thus subsume arbitrary first-order constraints). However, such constraints have not found their way into practical DBMS implementations yet and are unlikely to do so in the near future. In fact, most systems allow only restricted versions of FDs and INDs in the form of key dependencies and foreign key constraints, resp.

Definition 1. *Given a database instance r of R and a set of integrity constraints IC , we say that r is consistent if $r \models IC$ in the standard model-theoretic sense; inconsistent otherwise.*

We assume that we are dealing with *satisfiable* sets of constraints.

2.2 Repairs

Given a database instance r , the *set $\Sigma(r)$ of facts of r* is the set of ground facts $\{P(\bar{a}) \mid r \models P(\bar{a})\}$, where P is a relation name and \bar{a} a ground tuple. (There is clearly a straightforward correspondence between r and $\Sigma(r)$. However, we will find it more convenient to talk about the set of facts true in a first-order structure than about the structure itself.)

Definition 2. *The distance $\Delta(r, r')$ between database instances r and r' is defined as the symmetric difference of r and r' :*

$$\Delta(r, r') = (\Sigma(r) - \Sigma(r')) \cup (\Sigma(r') - \Sigma(r)).$$

Definition 3. *For the instances r, r', r'' , $r' \leq_r r''$ if $\Delta(r, r') \subseteq \Delta(r, r'')$, i.e., if the distance between r and r' is less than or equal to the distance between r and r'' .*

Definition 4. *Given a set of integrity constraints IC and database instances r and r' , we say that r' is a repair of r w.r.t. IC if $r' \models IC$ and r' is \leq_r -minimal in the class of database instances that satisfy IC .*

We denote by $\text{Repairs}_{IC}(r)$ the set of repairs of r w.r.t. IC . This set is nonempty because IC is satisfiable.

We will study in Section 4 some notions of repair that differ from that in Definition 4. Also, there is clearly a connection between the above notion of repair and the concepts of belief revision [42]. We discuss this connection in Section 6.

2.3 Queries

Queries are formulas over the same language \mathcal{L} as the integrity constraints. A query is *closed* (or a *sentence*) if it has no free variables. A closed query without quantifiers is also called *ground*. *Conjunctive queries* [23,2] are queries of the form

$$\exists \bar{x}_1, \dots, \bar{x}_m. (P_1(\bar{x}_1) \wedge \dots \wedge P_m(\bar{x}_m) \wedge \varphi(\bar{x}_1, \dots, \bar{x}_m))$$

where the variables of x_i are disjoint from that of x_j if $i \neq j$, and $\varphi(\bar{x}_1, \dots, \bar{x}_m)$ is a conjunction of built-in atomic formulas. A conjunctive query is *simple* if it has no repeated relation symbols and φ is of the form $c_1(\bar{x}_1) \wedge \dots \wedge c_m(\bar{x}_m)$.

The following definition is standard [2]:

Definition 5. A tuple \bar{t} is an answer to a query $Q(\bar{x})$ in an instance r iff $r \models Q(\bar{t})$.

2.4 Consistent query answers

Given a query $Q(\bar{x})$ to r , we want as *consistent* answers those tuples that are unaffected by the violations of *IC*, even when r violates *IC*.

Definition 6. [4] A tuple \bar{t} is a consistent answer to a query $Q(\bar{x})$ in a database instance r w.r.t. a set of integrity constraints *IC* iff \bar{t} is an answer to query $Q(\bar{x})$ in every repair r' of r w.r.t. *IC*. An \mathcal{L} -sentence Q is consistently true in r w.r.t. *IC* if it is true in every repair of r w.r.t. *IC*. In symbols:

$$r \models_{IC} Q(\bar{t}) \iff r' \models Q(\bar{t}) \text{ for every repair } r' \text{ of } r \text{ w.r.t. } IC.$$

Note: If the set of integrity constraints *IC* is clear from the context, we omit it for simplicity.

2.5 Examples

Example 1. Consider the following instance of a relation *Person*

<i>Name</i>	<i>City</i>	<i>Street</i>
Brown	Amherst	115 Klein
Brown	Amherst	120 Maple
Green	Clarence	4000 Transit

and the functional dependency $Name \rightarrow City \text{ Street}$. Clearly, the above instance does not satisfy the dependency. There are two repairs: one is obtained by removing the first tuple, the other by removing the second. The consistent answer to the query $Person(n, c, s)$ is just the tuple (Green, Clarence, 4000 Transit). On the other hand, the query $\exists s. Person(n, c, s)$ has two consistent answers: (Brown, Amherst) and (Green, Clarence). Similarly, the query

$$Person(\text{Brown}, \text{Amherst}, 115 \text{ Klein}) \vee Person(\text{Brown}, \text{Amherst}, 120 \text{ Maple})$$

is consistently true in the given instance of *Person*. Notice that for the last two queries the approach based on removing all inconsistent tuples and evaluating the original query using the remaining tuples gives different, less informative results.

Example 2. We give here some examples of denial constraints. Consider the relation *Emp* with attributes *Name*, *Salary*, and *Manager*, with *Name* being the primary key. The constraint that *no employee can have a salary greater than that of her manager* is a denial constraint:

$$\forall n, s, m, s', m'. (\neg Emp(n, s, m) \vee \neg Emp(m, s', m') \vee s \leq s').$$

Similarly, single-tuple constraints (CHECK constraints in SQL2) are a special case of denial constraints. For example, the constraint that *no employee can have a salary over \$200000* is expressed as:

$$\forall n, s, m. (\neg Emp(n, s, m) \vee s \leq 200000).$$

Note that a single-tuple constraint always leads to a single repair which consists of all the tuples of the original instance that satisfy the constraint.

2.6 Computational Problems

We consider here the following complexity classes:

- *P*: the class of decision problems solvable in polynomial time by deterministic Turing machines;
- *NP*: the class of decision problems solvable in polynomial time by nondeterministic Turing machines;
- *co-NP*: the class of decision problems whose complements are solvable in *NP*;
- Σ_2^P : the class of decision problems solvable in polynomial time by nondeterministic Turing machines with an *NP* oracle;
- Π_2^P : the class of decision problems whose complements are solvable in Σ_2^P ;
- AC^0 : the class of decision problems solvable by constant-depth, polynomial-size, unbounded fan-in circuits ($AC^0 \subset P$).

Assume a class of databases \mathcal{D} , a class of queries \mathcal{Q} and a class of integrity constraints \mathcal{C} are given. We study here the complexity of the following problems:

- *repair checking*, i.e., the complexity of the set

$$B_{IC} = \{(r, r') : r, r' \in \mathcal{D} \wedge r' \in Repairs_{IC}(r)\},$$

- *consistent query answers*, i.e., the complexity of the set

$$D_{IC, \Phi} = \{r : r \in \mathcal{D} \wedge r \models_{IC} \Phi\},$$

for a fixed sentence $\Phi \in \mathcal{Q}$ and a fixed finite set $IC \in \mathcal{C}$ of integrity constraints. This formulation is called *data complexity* (introduced by Chandra and Harel [24] and Vardi [63]), because it captures the complexity of a problem as a function of the number of tuples in the database instance only. The database schema, the query and the integrity constraints are assumed to be fixed. (This is not the only relevant notion of complexity. Under *combined complexity*, also introduced by Vardi [63], the constraints and the query are also part of the input. However, in the database context the consensus is that data complexity is of paramount importance, because the size of the database is typically several orders of magnitude larger than the size of the constraints and the query. Thus, data complexity reflects the real computational cost of evaluating a query more faithfully than combined complexity. Consequently, almost all existing results about the complexity of computing consistent query answers are about data complexity, the work of Cali, Lembo, and Rosati [19] being an exception.)

It is easy to see that even under a single key FD, there may be exponentially many repairs and thus the approach to computing consistent query answers by generating and examining all repairs is not feasible.

Example 3. Consider the functional dependency $A \rightarrow B$ and the following family of relation instances r_n , $n > 0$, each of which has $2n$ tuples (represented as columns) and 2^n repairs:

r_n								
A	a_1	a_1	a_2	a_2	\cdots	a_n	a_n	
B	b_0	b_1	b_0	b_1	\cdots	b_0	b_1	

On the other hand, Definitions 4 and 6 immediately yield the following result.

Proposition 1. *For every set of universal constraints F and \mathcal{L} -sentence Φ , B_F is in *co-NP* and $D_{F,\Phi}$ is in Π_2^P .*

Proof. An instance r' is not a repair of r if it violates the integrity constraints or there is another instance r'' which satisfies the constraints and is closer (in the sense of \leq_r) to r than r' . The first condition can be checked in P ; the second, in NP .

A sentence Φ is not consistently true in r if there is a repair r' of r in which Φ is false. Any repair r' of r can only contain tuples with the same constants as those occurring in the tuples of r . Thus the size of r' is polynomially bounded. Therefore, checking if there is a repair r' of r in which Φ is false can be done in NP with an NP oracle, i.e., it is in Σ_2^P .

In fact, B_F is in *co-NP* not only for universal but also for arbitrary first-order constraints. On the other hand, for non-universal constraints (e.g., INDs), the size of a repair cannot always be bounded and thus even the decidability of $D_{F,\Phi}$ is not guaranteed.

3 Denial constraints

A distinctive property of denial constraints is that their violations can only be removed by deleting tuples from the database. Therefore, repairs (in the sense of Definition 4) are always *subsets* of the original database. This property has a positive influence on the computational complexity of integrity maintenance. Most of the notions of repair that differ for broader classes of constraints (see Section 4) coincide in the case of denial constraints.

3.1 Query rewriting

Query rewriting is based on the following idea: *Given a query Q and a set of integrity constraints, construct a query Q' such that for every database instance r the set of answers to Q' in r is equal to the set of consistent answers to Q in r .*

Query rewriting was first proposed by Arenas, Bertossi, and Chomicki [4] in the context of domain relational calculus. The approach presented there was based on concepts from semantic query optimization (Chakravathy, Grant, and Minker [22]), in particular the notion of a *residue*.

Residues are associated with literals of the form $P(\bar{x})$ or $\neg P(\bar{x})$ (where \bar{x} is a vector of different variables of appropriate arity). For each literal $P(\bar{x})$ and each constraint containing $\neg P(\bar{x})$ in its clausal form (possibly after variable renaming), a local residue is obtained by removing $\neg P(\bar{x})$ and the quantifiers for \bar{x} from the (renamed) constraint. For each literal $\neg P(\bar{x})$ and each constraint containing $P(\bar{x})$ in its clausal form (possibly after variable renaming), a local residue is obtained by removing $P(\bar{x})$ and the quantifiers for the variables occurring only in \bar{x} from the (renamed) constraint. Finally, for each literal the global residue is computed as the conjunction of all local residues (possibly after normalizing variables).

Intuitively, the residues of a literal represent the conditions that must be true if the literal is true (and thus its negation is false).

Example 4. The constraint

$$\forall n, s, m, s', m'. (\neg Emp(n, s, m) \vee \neg Emp(m, s', m') \vee s \leq s')$$

produces for $Emp(n, s, m)$ the following local residues:

$$\forall s', m'. (\neg Emp(m, s', m') \vee s \leq s')$$

and

$$\forall s', m'. (\neg Emp(m', s', n) \vee s' \leq s).$$

The global residue is the conjunction of the local residues.

We consider queries that are conjunctions of positive and negative literals. The rewritten query is obtained in two steps. First, for every literal, an expanded version is constructed as the conjunction of this literal and its global residue. Second, the literals in the query are replaced by their expanded versions.

Example 5. Under the constraint

$$\forall n, s, m, s', m'. (\neg \text{Emp}(n, s, m) \vee \neg \text{Emp}(m, s', m') \vee s \leq s'),$$

the query $\text{Emp}(n, s, m)$ is rewritten into

$$\begin{aligned} & \text{Emp}(n, s, m) \wedge \forall s', m'. (\neg \text{Emp}(m, s', m') \vee s \leq s') \\ & \wedge \forall s', m'. (\neg \text{Emp}(m', s', n) \vee s' \leq s). \end{aligned}$$

A set of constraints is *generic* if it does not imply any ground literal. The results by Arenas, Bertossi, and Chomicki [4] imply the following:

Proposition 2. *For every generic set F of binary denial constraints and quantifier-free \mathcal{L} -sentence $\Phi(\bar{t})$ where*

$$\Phi = P_1(\bar{x}_1) \wedge \dots \wedge P_m(\bar{x}_m) \wedge \neg P_{m+1}(\bar{x}_{m+1}) \wedge \dots \wedge \neg P_n(\bar{x}_n) \wedge \varphi(\bar{x}_1, \dots, \bar{x}_n)$$

and \bar{t} is a vector of constants, $D_{F, \Phi(\bar{t})}$ is in P .

This is because query rewriting is done in a database-independent way and thus does not increase data complexity. In fact, since data complexity of evaluating first-order queries is in AC_0 , so is $D_{F, \Phi}$.

The paper [4] shows that query rewriting is also applicable to full inclusion dependencies. But then the literal expansion step might need to be iterated until no changes occur. Also, care needs to be taken about the termination of the expansion (this is addressed by Celle and Bertossi [21]). However, query rewriting does not generalize to non-binary constraints, general INDs, or queries involving disjunction or quantifiers. The latter is particularly disappointing, since disjunctions or existential quantifiers in queries are necessary to extract *partial information* from the database. For example, in Example 1, we would like to be able to derive from the database that Brown lives in Amherst at 115 Klein or 120 Maple. Moreover, non-binary constraints and disjunctions do not necessarily lead to intractability, as shown below.

3.2 Conflict hypergraph

Given a set of denial constraints F and an instance r , all the repairs of r with respect to F can be succinctly represented as the *conflict hypergraph*. This is a generalization of the *conflict graph* defined by Arenas, Bertossi, and Chomicki [6] for FDs only.

Definition 7. *The conflict hypergraph $\mathcal{G}_{F,r}$ is a hypergraph whose set of vertices is the set $\Sigma(r)$ of facts of an instance r and whose set of edges consists of all the sets*

$$\{P_1(\bar{t}_1), P_2(\bar{t}_2), \dots, P_l(\bar{t}_l)\}$$

such that $P_1(\bar{t}_1), P_2(\bar{t}_2), \dots, P_l(\bar{t}_l) \in \Sigma(r)$, and there is a constraint

$$\forall \bar{x}_1, \bar{x}_2, \dots, \bar{x}_l. (\neg P_1(\bar{x}_1) \vee \neg P_2(\bar{x}_2) \vee \dots \vee \neg P_l(\bar{x}_l) \vee \neg \varphi(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_l))$$

in F such that $P_1(\bar{t}_1), P_2(\bar{t}_2), \dots, P_l(\bar{t}_l)$ violate together this constraint, which means that there exists a substitution ρ such that $\rho(\bar{x}_1) = \bar{t}_1, \rho(\bar{x}_2) = \bar{t}_2, \dots, \rho(\bar{x}_l) = \bar{t}_l$ and $\varphi(\bar{t}_1, \bar{t}_2, \dots, \bar{t}_l)$ is true.

Note that there may be edges in $\mathcal{G}_{F,r}$ that contain only one vertex. Also, the size of the conflict hypergraph is polynomial in the number of tuples in the database instance.

Example 6. The conflict graph of the instance of the relation *Emp* from Example 1 is represented in Figure 1.

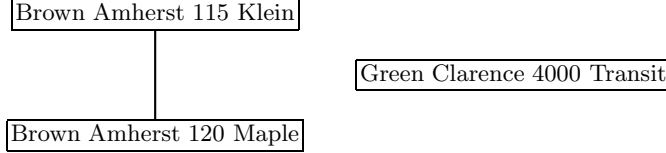


Fig. 1. Conflict hypergraph

By an *independent set* in a hypergraph we mean a subset of its set of vertices which does not contain any edge.

Proposition 3. *For each repair r' of r w.r.t. F , $\Sigma(r')$ is a maximal independent set in $\mathcal{G}_{F,r}$, and vice versa.*

Proof. Take an instance r' such that $\Sigma(r')$ is not a maximal independent set in $\mathcal{G}_{F,r}$. Then it contains an edge or is not maximal. In the first case, it means that r' violates the constraints; in the second, that r' is not minimal in \leq_r . On the other hand, if r' is not a repair, then it violates the constraints or is not minimal in \leq_r . In both cases, $\Sigma(r')$ is not a maximal independent set.

Proposition 3 yields the following result:

Proposition 4. [8] *For every set of denial constraints F and \mathcal{L} -sentence Φ , B_F is in P and $D_{F,\Phi}$ is in $co-NP$.*

Proof. Checking whether r' satisfies F is in P . The repair r' has also to be a maximal subset of r' that satisfies F . Checking that property can be done as follows: try all the tuples \bar{t} in $r - r'$, one by one. If $r' \cup \{\bar{t}\}$ satisfies F , then r' is not maximal. Otherwise, if for no such tuple \bar{t} , $r' \cup \{\bar{t}\}$ satisfies F , no superset of r' can satisfy F (violations of denial constraints cannot be removed by adding tuples) and r' is maximal. The fact that $D_{F,\Phi}$ is in $co-NP$ follows immediately from the definition of consistent query answer.

Note that the repairs of an instance r can be computed nondeterministically by picking a vertex of $\mathcal{G}_{F,r}$ which does not belong to a single-vertex edge and adding vertices that do not result in the addition of an entire edge.

3.3 Positive results

Theorem 1. [25,26] *For every set F of denial constraints and a ground sentence Φ , $D_{F,\Phi}$ is in P .*

Proof. We assume the sentence is in CNF³, i.e., of the form $\Phi = \Phi_1 \wedge \Phi_2 \wedge \dots \wedge \Phi_l$, where each Φ_i is a disjunction of ground literals. Φ is true in every repair of r if and only if each of the clauses Φ_i is true in every repair. So it is enough to provide a polynomial algorithm which will check if a given ground clause is consistently true in an instance r .

It is easier to think that we are checking if a ground clause is **not** consistently true in r . This means that we are checking, whether there exists a repair r' in which $\neg\Phi_i$ is true for some i . But $\neg\Phi_i$ is of the form

$$P_1(\bar{t}_1) \wedge P_2(\bar{t}_2) \wedge \dots \wedge P_m(\bar{t}_m) \wedge \neg P_{m+1}(\bar{t}_{m+1}) \wedge \dots \wedge \neg P_n(\bar{t}_n),$$

where the \bar{t}_j 's are tuples of constants. WLOG, we assume that all the facts in the set $\{P_1(\bar{t}_1), \dots, P_n(\bar{t}_n)\}$ are mutually distinct.

The nonderministic algorithm selects for every j , $m+1 \leq j \leq n$, $P_j(\bar{t}_j) \in \Sigma(r)$, an edge $E_j \in \mathcal{G}_{F,r}$ such that $P_j(\bar{t}_j) \in E_j$, and constructs a set of facts S such that

$$S = \{P_1(\bar{t}_1), \dots, P_m(\bar{t}_m)\} \cup \bigcup_{m+1 \leq j \leq n, P_j(\bar{t}_j) \in \Sigma(r)} (E_j - \{P_j(\bar{t}_j)\})$$

and *there is no edge $E \in \mathcal{G}_{F,r}$ such that $E \subseteq S$* . If the construction of S succeeds, then a repair in which $\neg\Phi_i$ is true can be built by adding to S new facts from $\Sigma(r)$ until the set is maximal independent. The algorithm needs $n - m$ nondeterministic steps, a number which is independent of the size of the database (but dependent on Φ), and in each of its nondeterministic steps selects one possibility from a set whose size is polynomial in the size of the database. So there is an equivalent polynomial-time deterministic algorithm.

In the case when the set F of integrity constraints consists of only one FD per relation the conflict hypergraph has a very simple form. It is a disjoint union of full multipartite graphs. If this single dependency is a key dependency, then the conflict graph is a union of disjoint cliques. Because of this very simple structure we hoped that it would be possible, in such a situation, to compute in polynomial time the consistent answers not only to ground queries, but also to all conjunctive queries. As we are going to see now, this is only possible if the conjunctive queries are suitably restricted.

Theorem 2. [25,26] *Let F be a set of FDs, each dependency over a different relation among P_1, P_2, \dots, P_k . Then for each closed simple conjunctive query Q , there exists a closed query Q' such that for every database instance r , $r \models_F Q$ iff $r \models Q'$. Consequently, $D_{F,Q}$ is in P .*

³ This assumption does not reduce the generality of our results, because every ground query can be converted to CNF independently of the database, and thus without affecting the data complexity of query evaluation. However, from a practical point of view, CNF conversion may lead to unacceptably complex queries.

Proof. We present the construction for $k = 2$ for simplicity; the generalization to an arbitrary k is straightforward. Let P_1 and P_2 be two different relations of arity k_1 and k_2 , resp. Assume we have the following FDs: $Y_1 \rightarrow Z_1$ over P_1 and $Y_2 \rightarrow Z_2$ over P_2 . Let \bar{y}_1 be a vector of arity $|Y_1|$, \bar{y}_2 a vector of arity $|Y_2|$, \bar{z}_1 and \bar{z}'_1 vectors of arity $|Z_1|$, and \bar{z}_2 and \bar{z}'_2 vectors of arity $|Z_2|$. Finally, let $\bar{w}_1, \bar{w}'_1, \bar{w}''_1$ (resp. $\bar{w}_2, \bar{w}'_2, \bar{w}''_2$) be vectors of arity $k_1 - |Y_1| - |Z_1|$ (resp. $k_2 - |Y_2| - |Z_2|$). All of the above vectors consist of distinct variables. The query Q is of the following form

$$\exists \bar{y}_1, \bar{z}_1, \bar{w}_1, \bar{y}_2, \bar{z}_2, \bar{w}_2. (P_1(\bar{y}_1, \bar{z}_1, \bar{w}_1) \wedge P_2(\bar{y}_2, \bar{z}_2, \bar{w}_2) \wedge c_1(\bar{y}_1, \bar{z}_1, \bar{w}_1) \wedge c_2(\bar{y}_2, \bar{z}_2, \bar{w}_2)).$$

Then, the query Q' (which is also closed) is as follows:

$$\begin{aligned} & \exists \bar{y}_1, \bar{z}_1, \bar{w}_1, \bar{y}_2, \bar{z}_2, \bar{w}_2 \forall \bar{z}'_1, \bar{w}'_1, \bar{z}'_2, \bar{w}'_2 \exists \bar{w}''_1, \bar{w}''_2. (P_1(\bar{y}_1, \bar{z}_1, \bar{w}_1) \wedge P_2(\bar{y}_2, \bar{z}_2, \bar{w}_2) \\ & \wedge c_1(\bar{y}_1, \bar{z}_1, \bar{w}_1) \wedge c_2(\bar{y}_2, \bar{z}_2, \bar{w}_2) \wedge (P_1(\bar{y}_1, \bar{z}'_1, \bar{w}'_1) \wedge P_2(\bar{y}_2, \bar{z}'_2, \bar{w}'_2) \Rightarrow \\ & P_1(\bar{y}_1, \bar{z}'_1, \bar{w}''_1) \wedge P_2(\bar{y}_2, \bar{z}'_2, \bar{w}''_2) \wedge c_1(\bar{y}_1, \bar{z}'_1, \bar{w}''_1) \wedge c_2(\bar{y}_2, \bar{z}'_2, \bar{w}''_2)). \end{aligned}$$

Chomicki and Marcinkowski [26] describe the generalizations of Theorems 1 and 2 to the non-ground case. The generalized version of the algorithm presented in the proof of Theorem 1 has been implemented as a part of a database middleware system Hippo [27,28].

The above line of research is continued by Fuxman and Miller [41]. They make the observation that the query

$$Q_1 \equiv \exists e_1, e_2, s. (R(e_1, s) \wedge R(e_2, s) \wedge e_1 \neq e_2)$$

is consistently false if and only if there is a perfect matching in the graph of the relation R (the first argument of R is understood to be the key here). This implies that Q_1 is an example of a tractable query which cannot be answered by query rewriting (see the discussion earlier in this section), because perfect matching is not first order definable. Fuxman and Miller [41] generalize this perfect matching technique to a wider class of queries on databases over binary schemata.

3.4 Negative results

We show now that Theorems 1 and 2 are the strongest possible (assuming the arity of relations is not restricted), because relaxing any of their conditions leads to *co-NP-completeness*. This is the case even though we limit ourselves to *key* FDs.

Theorem 3. [25,26] *There exist a key FD f and a closed conjunctive query*

$$Q \equiv \exists x, y, y', z. (R(x, y, c) \wedge R(z, y', d) \wedge y = y'),$$

for which $D_{\{f\}, Q}$ is *co-NP-complete*.

Proof. Reduction from MONOTONE 3-SAT. The FD is $A \rightarrow BC$. Let $\beta = \phi_1 \wedge \dots \wedge \phi_m \wedge \psi_{m+1} \dots \wedge \psi_l$ be a conjunction of clauses, such that all occurrences of variables in ϕ_i are positive and all occurrences of variables in ψ_i are negative. We build a database r_β with the facts $R(i, p, c)$ if the variable p occurs in the clause ϕ_i and $R(i, p, d)$ if the variable p occurs in the clause ψ_i . Now, there is an assignment which satisfies β if and only if there exists a repair of the database r_β in which Q is false.

As an example, consider the following monotone formula

$$\beta_0 = s \wedge (p \vee q) \wedge \neg p \wedge (\neg q \vee \neg s).$$

The corresponding database r_{β_0} contains the following facts:

$$R(1, s, c), R(2, p, c), R(2, q, c), R(3, p, d), R(4, q, d), R(4, s, d).$$

Clearly, β_0 is unsatisfiable. Also, Q is true in every repair of r_{β_0} .

To show the \Rightarrow implication, select for each clause ϕ_i one variable p_i which occurs in this clause and whose value is 1 and for each clause ψ_i , one variable p_i which occurs in ψ_i and whose value is 0. The set of facts $\{R(i, p_i, c) : i \leq m\} \cup \{R(i, p_i, d) : m+1 \leq i \leq l\}$ is a repair in which the query Q is false. The \Leftarrow implication is even simpler.

Note that the query in Theorem 3 is nonsimple.

Theorem 4. [25,26] *There is a set F of two key dependencies and a closed conjunctive query $Q \equiv \exists x, y. R(x, y, b)$, for which $D_{F,Q}$ is co-NP-complete.*

The above result was obtained first in a slightly weaker form – for non-key FDs – in [6,8].

Theorem 5. [25,26] *There exist a denial constraint f and a closed conjunctive query $Q \equiv \exists x, y. R(x, y, b)$, for which $D_{\{f\},Q}$ is co-NP-complete.*

For the *co-NP-hard* cases identified above, no approach to the computation of consistent query answers based on query rewriting can work. This is because (a) such approaches produce queries that are evaluable in AC^0 , and (b) the relevant *co-NP-complete* problems are not in AC^0 .

4 Beyond denial constraints

4.1 Different notions of repair

The basic notion of repair (Definition 4) requires that the *symmetric* difference between a database and its repair is minimized. But as we have seen, in the context of denial constraints every repair is a subset of the database. Thus, insertions are not used in constructing repairs. The situation is different for more general constraints.

Example 7. Consider the integrity constraint $\forall x. (\neg P(x) \vee R(x))$ and the database $\{P(1)\}$. Then there are two repairs: \emptyset and $\{P(1), R(1)\}$.

Allowing repairs constructed using insertions makes sense if the information in the database may be incomplete⁴. The latter is common in data integration applications where the data is pulled from multiple sources, typically without any guarantees on its completeness. On the other hand, if we know that the data in the database is complete but possibly incorrect, as in data warehousing applications, it is natural to consider only repairs constructed using deletions. Current language standards like SQL:1999 [60] allow only deletions in their repertoire of referential integrity actions. Moreover, the restriction to deletions guarantees that the number of repairs is finite. This restriction is also beneficial from the computational point of view, as we will see later. Note that the symmetric restriction to insertions would make sense only in the context of inclusion dependencies: denial constraints cannot be fixed by insertions.

Example 8. Consider a database with two relations $Employee(SSN, Name)$ and $Manager(SSN)$. There are functional dependencies $SSN \rightarrow Name$ and $Name \rightarrow SSN$, and an inclusion dependency $Manager[SSN] \subseteq Employee[SSN]$. The relations have the following instances:

<i>Employee</i>	
<i>SSN</i>	<i>Name</i>
123456789	Smith
555555555	Jones
555555555	Smith

<i>Manager</i>
<i>SSN</i>
123456789
555555555

The instances do not violate the IND but violate both FDs. If we consider only the FDs, there are two repairs: one obtained by removing the third tuple from *Employee*, and the other by removing the first two tuples from the same relation. However, the second repair violates the IND. This can be fixed by removing the first tuple from *Manager*. So if we consider all the constraints, there are two deletion-only repairs:

<i>Employee</i>	
<i>SSN</i>	<i>Name</i>
123456789	Smith
555555555	Jones

<i>Manager</i>
<i>SSN</i>
123456789
555555555

and

<i>Employee</i>	
<i>SSN</i>	<i>Name</i>
555555555	Smith

<i>Manager</i>
<i>SSN</i>
555555555

⁴ *Incompleteness* here does not mean that the database contains *indefinite information* in the form of nulls or disjunctions [61]. Rather, it means that *Open World Assumption* is adopted, i.e., the facts missing from the database are not assumed to be false.

Finally, insertions may lead to infinitely many repairs of the form

<i>Employee</i>	
<i>SSN</i>	<i>Name</i>
123456789	c
555555555	Smith

<i>Manager</i>
<i>SSN</i>
123456789
555555555

where c is an arbitrary string different from *Smith* (this is forced by one of the FDs).

Example 9. To see the plausibility of repairing by insertion, consider Example 1 again. Suppose the only constraint is that *City* is a foreign-key, referencing the key of another relation *Cities*. Then if Clarence does not occur as a key value in the current instance of *Cities*, then one possible repair is to delete the tuple $\langle \text{Green, Clarence, 4000 Transit} \rangle$ (the city name may be erroneous). But it is also possible that not all the cities are in the current instance of *Cities*, thus another way of resolving the inconsistency is to insert a tuple with city name = Clarence into *Cities*. If *Cities* has more than one attribute, then infinitely many such repairs arise (all possible values for the nonkey attributes have to be considered). In practice, the nonkey attributes will receive null values in such a case.

In the next subsection, we will consider deletion-only repairs. Afterwards, we will revert to Definition 4.

4.2 Repairing by deletion

We modify here Definition 4 to allow only repairs obtained by deletion of one or more tuples from the original instance. The definition of consistent query answers remains unchanged.

We also use B_I^- (resp. $D_{I,\phi}^-$) to denote the problem of repair of repair checking (resp. consistent query answers), in order to indicate that due to the different notion of repair those problems are different from the corresponding problems B_I and $D_{I,\phi}$ studied earlier.

We establish first a general relationship between the problems of repair checking and consistent query answers.

Theorem 6. *In the presence of full foreign key constraints, the problem of repair checking is logspace-reducible to the complement of the problem of consistent query answers.*

Proof. We discuss here the case of the database consisting of a single relation R_0 . Assume r is the given instance of R_0 and r' is another instance of R_0 satisfying the set of integrity constraints IC . We define a new relation S_0 having the same attributes as R_0 plus an additional attribute Z . Consider an instance s of S_0 built as follows:

- for every tuple $(x_1, \dots, x_k) \in r'$, we add the tuple (x_1, \dots, x_k, c_1) to s ;

– for every tuple $(x_1, \dots, x_k) \in r - r'$, we add the tuple (x_1, \dots, x_k, c_2) to s .

Consider also another relation P having a single attribute W , and a foreign key constraint $i_0 : P[W] \subseteq S_0[Z]$. The instance p of P consists of a single tuple c_2 . We claim that $P(c_2)$ is consistently true in the database instance consisting of s and p w.r.t. $IC \cup \{i_0\}$ iff r' is not a repair of r w.r.t. IC .

We now characterize the computational complexity of repair checking and consistent query answers for FDs and INDs.

Proposition 5. *For every set of INDs I and \mathcal{L} -sentence Φ , B_I^- and $D_{I,\Phi}^-$ are in P .*

Proof. For a given database instance r , a single repair is obtained by deleting all the tuples violating I (and only those).

We consider now FDs and INDs together. We want to identify the cases where both repair checking and computing consistent query answers can be done in P . The intuition is to limit the interaction between the FDs and the INDs in the given set of integrity constraints in such a way that one can use the polynomial-time results presented earlier.

Lemma 1. [26] *Let $IC = F \cup I$ be a set of constraints consisting of a set of key FDs F and a set of foreign key constraints I but with no more than one key per relation. Let r be a database instance and r' be the unique repair of r with respect to the foreign key constraints in I . Then r'' is a repair of r w.r.t. IC if and only if it is a repair of r' w.r.t. F .*

Proof. The only thing to be noticed here is that repairing r' with respect to key constraints does not lead to new inclusion violations. This is because the set of key values in each relation remains unchanged after such a repair (which is not necessarily the case if we have relations with more than one key).

Corollary 1. *Under the assumptions of Lemma 1, B_{IC}^- is in P .*

The repairs w.r.t. $IC = F \cup I$ of r are computed by (deterministically) repairing r w.r.t. I and then nondeterministically repairing the result w.r.t. F (as described in the previous section).

We can also transfer the polynomial-time results about consistent query answers obtained for FDs only.

Corollary 2. *Let Φ a quantifier-free \mathcal{L} -sentence or a simple conjunctive closed \mathcal{L} -query. Then under the assumptions of Lemma 1, $D_{IC,\Phi}^-$ is in P .*

Unfortunately, the cases identified above are the only ones we know of in which both repair checking and consistent query answers are in P .

For acyclic INDs (and arbitrary FDs), the repair checking problem is still in P . Surprisingly, consistent query answers becomes in this case a *co-NP-hard* problem, even in the case of key FDs and primary key foreign key constraints. If we relax any of the assumptions of Lemma 1, the problem of consistent query answers becomes intractable, even under acyclicity.

Definition 8. [2] *Let I be a set of INDs over a database schema R . Consider a directed graph whose vertices are relations from R and such that there is an edge $E(P, R)$ in the graph if and only if there is an IND of the form $P[X] \subseteq R[Y]$ in I . A set of inclusion dependencies is acyclic if the above graph does not have a cycle.*

Theorem 7. [26] *Let $IC = F \cup I$ be a set of constraints consisting of a set of FDs F and an acyclic set of INDs I . Then B_{IC}^- is in P .*

Proof. First compare r and r' on relations which are not on the left-hand side of any IND in I . Here, r' is a repair if and only if the functional dependencies are satisfied in r' and if adding to it any additional tuple from r would violate one of the functional dependencies. Then consider relations which are on the left-hand side of some INDs, but the inclusions only lead to already checked relations. Again, r' is a repair of those relations if and only if adding any new tuple (i.e. any tuple from r but not from r') would violate some constraints. Repeat the last step until all the relations are checked.

The above proof yields a nondeterministic polynomial-time procedure for computing the repairs w.r.t. $IC = F \cup I$.

To our surprise, Theorem 7 is the strongest possible positive result. The problem of consistent query answers is already intractable, even under additional restrictions on the FDs and INDs.

Theorem 8. [26] *There exist a database schema, a set IC of integrity constraints consisting of key FDs and of an acyclic set of primary foreign key constraints, and a ground atomic query Φ such that $D_{IC, \Phi}^-$ is co-NP-hard.*

We show also that relaxing the acyclicity assumption in Theorem 7 leads to the intractability of the repair checking problem (and thus also the problem of consistent query answers), even though alternative restrictions on the integrity constraints are imposed.

Theorem 9. [26] *There exist a database schema and a set IC of integrity constraints, consisting of one FD and one IND, such that B_{IC}^- is co-NP-hard.*

Theorem 10. [26] *There exist a database schema and a set IC of integrity constraints, consisting of key FDs and foreign key constraints, such that B_{IC}^- is co-NP-hard.*

We complete the picture by considering arbitrary FDs and INDs.

Theorem 11. [26] *For an arbitrary set IC of FDs and INDs, B_{IC}^- is co-NP-complete.*

Theorem 12. [26] *For an arbitrary set IC of FDs and INDs, and quantifier-free \mathcal{L} -sentence Φ , $D_{IC, \Phi}^-$ is Π_2^P -complete.*

Proof. The membership in Π_2^P follows from the definition of consistent query answer. We show Π_2^P -hardness below.

Consider a quantified boolean formula β of the form

$$\beta \equiv \forall p_1, p_2, \dots, p_k \exists q_1, q_2, \dots, q_l \psi$$

where ψ is quantifier-free and equals to $\psi_1 \wedge \psi_2 \wedge \dots \wedge \psi_m$, where ψ_i are clauses. We will construct a database instance r_β , over a schema with a single relation $R(A, B, C, D)$, such that $R(a, a, \psi_1, a)$ is a consistent answer if and only if β is true. The integrity constraints will be $A \rightarrow B$ and $C \subseteq D$.

There are 3 kinds of tuples in r_β . For each occurrence of a literal in ψ we have one tuple of the first kind (we adopt the convention that ψ_{m+1} is ψ_1):

- $R(p_i, 1, \psi_j, \psi_{j+1})$ if p_i occurs positively in ψ_j ,
- $R(q_i, 1, \psi_j, \psi_{j+1})$ if q_i occurs positively in ψ_j ,
- $R(p_i, 0, \psi_j, \psi_{j+1})$ if p_i occurs negatively in ψ_j ,
- $R(q_i, 0, \psi_j, \psi_{j+1})$ if q_i occurs negatively in ψ_j .

For each universally quantified variable p_i we have two tuples of the second kind: $R(p_i, 1, a_i, a_i)$ and $R(p_i, 0, a_i, a_i)$. Finally, there is just one tuple of the third kind: $R(a, a, \psi_1, a)$.

Consider a repair s of r_β . Call s *white* if it does not contain any tuple of the first kind. Call s *black* if for each clause ψ_i of ψ , s contains some tuple of the form $R(-, -, \psi_i, \psi_{i+1})$. We claim, that each repair of r_β is either white or black. Indeed, if some $R(-, -, \psi_j, \psi_{j+1})$ is in s (i.e. if s is not white) then, since the $C \subseteq D$ constraint is satisfied in s , there must be some tuple of the form $R(-, -, \psi_{j-1}, \psi_j)$ in s . But the last implies that also some $R(-, -, \psi_{j-2}, \psi_{j-1})$ must be in s , and so on.

Notice, that it follows from the $C \subseteq D$ constraint that if a repair s is white, then $R(a, a, \psi_1, a)$ cannot be in s . On the other hand, it is easy to see that if s is black, then $R(a, a, \psi_1, a)$ is in s .

Now, for a repair s of r_β define σ_s^1 (respectively σ_s^2) as the substitution resulting from projecting the set of the tuples of the first (resp. second) kind in s on the first two attributes. Notice that σ_s^1 and σ_s^2 agree on the shared arguments: this is since s satisfies the functional dependency. From the construction of r_β it follows that if s is black then $\sigma_s^1(\psi)$ is true (for each ψ_j there is either a variable x occurring positively in ψ , such that $\sigma_s^1(x) = 1$ or variable x occurring negatively in ψ , such that $\sigma_s^1(x) = 0$).

To end the proof we need to show that β is false if and only if there exists some white repair of r_β .

Suppose β is false. Let σ be such a valuation of the variables p_1, p_2, \dots, p_k that the formula $\sigma(\beta)$ (with free variables q_1, q_2, \dots, q_l) is not satisfiable. The set s_σ of all the tuples from r_β which are of the form $R(p_i, \sigma(p_i), a_i, a_i)$ is consistent. So there exists a repair s such that $s_\sigma \subseteq s$. But if s is black then σ_s^1 is a substitution which agrees with σ and satisfies ψ , which is a contradiction. So s must be white.

For the opposite direction, suppose β is true, and s is some white repair of r_β . This means that s contains only tuples of the second kind, and the projection

of s on the first two attributes is some valuation σ of the variables p_1, p_2, \dots, p_k . Since β is true, there exists a valuation σ' of the variables q_1, q_2, \dots, q_l such that $\sigma'\sigma(\psi)$ is true. Now, the union of s and the set of all the tuples of the first kind which are of the form $R(p_i, \sigma(p_i), \psi_j, \psi_{j+1})$ or of the form $R(q_i, \sigma'(q_i), \psi_j, \psi_{j+1})$ is a consistent superset of s , which contradicts the assumption that s was a repair.

4.3 Repairing by insertion and deletion

Calì, Lembo, and Rosati [19] study the complexity of query answering when the database is possibly not only inconsistent but also incomplete. Like in the paper of Arenas, Bertossi, and Chomicki [4] and the follow-up work, consistency is defined by means of integrity constraints, and an answer to a query is understood to be consistent if it is true in all possible repairs. Six definitions of the notion of repair are considered by Calì, Lembo, and Rosati [19], only one of which coincides with Definition 4. Each of those notions postulates that a repair satisfy the integrity constraints.

The *sound*, *exact*, and *complete* semantics do not impose any minimality conditions on repairs. The sound semantics requires that a repair is a superset of the database; the exact semantics – that it is equal to the database; and the complete semantics – that it is a subset of the database. Because an empty database satisfies any set of FDs and INDs, it is a repair under the complete semantics. Therefore, in this case there is no nontrivial notion of consistent query answer. The exact semantics is uninteresting for a different reason: the set of repairs is empty if the database violates the constraints, and consists of the original database if the constraints are satisfied. The sound semantics is suitable if the constraints consist of INDs only; in the presence of FDs, the set of repairs may be empty (this is because the violations of FDs cannot be fixed by tuple insertions). However, solving a violation of an inclusion dependency by adding new tuples may lead to new violations, of other dependencies, and thus there is no clear upper bound on the size of a minimal repair, under the sound semantics. So one can expect the problem of consistent query answers to be undecidable here. And indeed, this undecidability is proved by Calì, Lembo, and Rosati [19].

To present the decidable cases identified by Calì, Lembo, and Rosati [19], we need to introduce some definitions.

Definition 9. *Let $IC = F \cup I$ be a set of constraints consisting of a set of key FDs F (with at most one key per relation) and a set of INDs I . Then an IND $P[X] \subseteq R[Y] \in I$ is non-key-conflicting w.r.t. F if either: (1) no nontrivial key FD is defined for R in F , or (2) Y is not a strict superset of the key of R .*

Theorem 13. [19] *Let $IC = F \cup I$ be a set of constraints consisting of a set of key FDs F (with at most one key per relation) and a set of non-key-conflicting INDs I . Let Q be a union of conjunctive queries. Then the problem of consistent query answers is in P (under sound semantics).*

Notice that Theorem 13 to some degree parallels Lemma 1. One has to bear in mind, however, that those results use different semantics of consistent query answers due to different notions of repair.

The notion of repair under the sound semantics is not powerful enough if some functional dependencies can be violated in the original database (the set of repairs may be empty). This observation leads to the notions of *loosely-complete*, *loosely-sound*, and *loosely-exact* semantics. Under those semantics, repairs are constructed by adding tuples, as well as by deleting them. The loosely-complete semantics does not impose the requirement that the set of deleted tuples be minimal in a repair; therefore, the empty database is a repair and, as under the complete semantics, the notion of consistent query answer is trivial. The notion of repair under the loosely-exact semantics is identical to that of Definition 4. Finally, loosely-sound semantics requires only that the set of deleted tuples is minimized.

Cali, Lembo, and Rosati [19] show that for general key FDs and INDs the problem of consistent query answers under loosely-sound and loosely-exact semantics is undecidable. The decidable cases identified in that paper involve again non-key-conflicting INDs.

Theorem 14. [19] *Let $IC = F \cup I$ be a set of constraints consisting of a set of key FDs F (with at most one key per relation) and a set of non-key-conflicting INDs I . Let Q be a union of conjunctive queries. Then the problem of consistent query answers is co-NP-complete (under loosely-sound semantics) and Π_2^P -complete (under loosely-exact semantics).*

Contrasting Theorem 14 with Theorem 13, we see that the loosely-sound semantics augments the sound semantics with the nondeterministic choice of the set of tuples to be deleted. The loosely-exact semantics adds another level of nondeterminism.

We conclude by noting that none of the six notions of repair coincides with the one proposed by Chomicki and Marcinkowski [26]. The latter notion, by forcing the repairs to be subsets of the original database, makes the problem of consistent query answers decidable (Theorem 12).

4.4 Repairing by attribute modification

In the framework of Arenas, Bertossi and Chomicki [4], which was adapted by the follow-up papers [5,6,7,8,9,11,12,19,21,25,26,34,41,45,46,49,62], the smallest unit to be deleted from (or added to) a database in the process of repairing is a tuple. A different choice is made by Wijzen [64] where tuples themselves are being repaired. The idea there is that even a tuple that violates the integrity constraints can possibly still yield some important information. Wijzen's motivating example is a relation of arity 5 containing information on dioxin levels in food samples. The attributes of this relation are: the sample number, the sample date, the analysis date, the lab and the dioxin level. The integrity constraint is "the sample date must be prior to the analysis date". This is a denial constraint, thus the

only thing that can be done with a tuple violating this constraint is dropping it, possibly getting rid of the number and other data of the sample, which may indicate an alarming dioxin level.

Example 10. Consider the relation *Emp* with attributes *Name*, *Salary*, and *Manager*, where *Name* is the primary key. The constraint that *no employee can have a salary greater than that of her manager* is a denial constraint:

$$\forall n, s, m, s', m'. (\neg Emp(n, s, m) \vee \neg Emp(m, s', m') \vee s \leq s').$$

Consider the following instance of *Emp* that violates the constraint:

<i>Name</i>	<i>Salary</i>	<i>Manager</i>
Jones	120K	Black
Black	100K	Black

Under Definition 4, this instance has two repairs: one obtained by deleting the first tuple and the other – by deleting the second tuple. It might be more natural to consider the repairs obtained by adjusting the individual salary values in such a way that the constraint is satisfied.

The basic idea of the approach proposed by Wijzen [64] is to define the notion of repair by means of the ordering defined by the *subsumption of tableaux*, instead of the ordering \leq_r defined by set inclusion (Definition 3). A tableau is a generalization of a relation: tuples can have not only constants but also variables as components. Wijzen [64] considers only single-relation databases.

Definition 10. [64] *If S and T are two tableaux, then:*

- S subsumes T ($S \succeq T$) *if there is a substitution σ such that $\sigma(T) \subseteq S$;*
- S one-one subsumes T ($S \sqsupseteq T$) *if there is a substitution σ such that $\sigma(T) \subseteq S$ and $|\sigma(T)| = |T|$.*

Definition 11. [64] *A tableau T subsatisfies a set of integrity constraints IC if there is a relation R satisfying IC and such that $R \succeq T$. A fix, with respect to IC , of a relation D is any tableau T such that*

- (i) $D \sqsupseteq T$ and T subsatisfies IC ;
- (ii) T is subsumption-maximal among tableaux satisfying (i).

A repair of D is now any minimal relation D_1 which satisfies IC and for which there exists a fix T of D such that $D_1 \succeq T$.

The notion of consistent query answer in Wijzen’s framework is that of Definition 6 in which the notion of repair of Definition 4 is substituted by that of Definition 11.

Example 11. If the dioxin database contains just one tuple

$$\langle 120, 17Jan2002, 16Jan2002, ICI, 150 \rangle,$$

then there are two fixes of it:

$$\langle 120, x, 16\text{Jan}2002, ICI, 150 \rangle$$

and

$$\langle 120, 17\text{Jan}2002, y, ICI, 150 \rangle.$$

A repair is any database resulting from the first fix by substituting for x any date prior to 16 Jan 2002 or from the second fix by substituting for y any date later than 17 Jan 2002. In each repair there is a tuple with the dioxin level 150.

The class of integrity constraints considered by Wijsen consists of tuple- and equality-generating dependencies [55,2]. The first are simply universal Horn constraints, the second – restricted denial constraints. The queries studied by Wijsen [64] are conjunctive queries.

Wijsen [64] considers cases where answers to conjunctive queries can be computed by means of *early repairs*, which means, that for a given relation D , another relation D' is computed, such that a query is consistently true in D if and only if the query is true in D' . There are questions left open by the paper regarding the size of D' , and in consequence, regarding the efficiency of this algorithm. Recently [Jef Wijsen, unpublished], *NP*-hardness of repair checking has been established under Definition 11 of repair:

Theorem 15. *There exists a denial constraint ϕ_0 with one database literal, such that repair checking is NP-hard.*

Note that for denial constraints repair checking under Definition 4 is in *P*. Thus, unless $P = NP$, there is a considerable computational price for using Wijsen's framework.

We note that the notions of repair discussed so far do not exhaust all the possibilities. For example, in Example 11 the database could also be repaired by swapping the values of the second and third attributes.

5 Aggregation

So far we have considered only first-order queries but in databases aggregation queries are also important. In fact, aggregation is essential in scenarios, like data warehousing, where inconsistencies are likely to occur and keeping inconsistent data may be useful.

We consider here a restricted scenario: there is only one relation and integrity constraints are limited to functional dependencies. Thus, every repair in the sense of Definition 4 is a maximal consistent subset of the given instance. Aggregation queries consist of single applications of one of the standard SQL-2 aggregation operators (*MIN*, *MAX*, *COUNT*(*), *COUNT*(A), *SUM*, and *AVG*). Even in this case, it was shown by Arenas, Bertossi and Chomicki [6] that computing consistent query answers to aggregation queries is a challenging problem for both semantic and complexity-theoretic reasons.

Example 12. Consider the following instance r of the relation Emp :

<i>Name</i>	<i>Salary</i>	<i>Manager</i>
Brown	50K	Black
Brown	70K	Black
Green	40K	Brown

It is inconsistent w.r.t. the FD: $Name \rightarrow Salary$. The repairs are:

<i>Name</i>	<i>Salary</i>	<i>Manager</i>
Brown	50K	Black
Green	40K	Brown

<i>Name</i>	<i>Salary</i>	<i>Manager</i>
Brown	70K	Black
Green	40K	Brown

If we pose the query

```
SELECT MIN(Salary) FROM Emp
```

we should get 40K as a consistent answer: $MIN(Salary)$ returns 40K in each repair. Nevertheless, if we ask

```
SELECT MAX(Salary) FROM Emp
```

then the maximum, 70K, comes from a tuple that participates in the violation of the FD. Actually, $MAX(Salary)$ returns a different value in each repair: 50K or 70K. Thus, there is no consistent answer in the sense of Definition 6. \square

We give a new, slightly weakened definition of consistent answer to an aggregation query that addresses the above difficulty.

Definition 12. [6] *Given a set of integrity constraints F , an aggregation query f and a database instance r , the set of possible answers $Poss_F^f(r)$ is defined as*

$$Poss_F^f(r) = \{f(r') \mid r' \in Repairs_F(r)\}.$$

The greatest-lower-bound (glb) answer $glb_F(f, r)$ to f w.r.t. F in r is defined as

$$glb_F(f, r) = glb Poss_F^f(r).$$

The least-upper-bound (lub) answer $lub_F(f, r)$ to f w.r.t. F in r is defined as

$$lub_F(f, r) = lub Poss_F^f(r).$$

\square

According to this definition, in Example 12, 50K (resp. 70K) are the *glb-answer* (resp. *lub-answer*) to the query

```
SELECT MAX(Salary) FROM Emp.
```

Notice that the interval $[glb\ answer, lub\ answer]$ represents in a *succinct* form a superset of the values that the aggregation query can take in all possible repairs of the database r w.r.t. a set of FDs. The representation of the interval is always polynomially sized, since the numeric values of the endpoints can be represented in binary.

Example 13. Along the lines of Example 3, consider the functional dependency $A \rightarrow B$ and the following family of relation instances S_n , $n > 0$:

r_n											
A	a_1	a_1	a_2	a_2	a_3	a_3	\cdots	a_n	a_n		
B	0	1	0	2	0	4	\cdots	0	2^n		

The aggregation query $\text{SUM}(B)$ takes all the exponentially many values between 0 and $2^{n+1} - 1$ in the (exponentially many) repairs of the database [6]. An explicit representation of the possible values the aggregation function would then be exponentially large. Moreover, it would violate the 1NF assumption. On the other hand, the glb/lub representation has polynomial size. \square

Arenas et al. [8] provide a complete classification of the tractable and intractable cases of the problem of computing consistent query answers (in the sense of Definition 12) to aggregation queries. Its results can be summarized as follows:

Theorem 16. [8] *The problem of computing glb/lub answers is in P for all the aggregate operators except $\text{COUNT}(A)$, if the set of integrity constraints contains at most one non-trivial FD.*

Theorem 17. [8] *The problem of checking whether the glb-answer to a query is $\leq k$ and the problem of checking whether the lub-answer to a query is $\geq k$ are NP-complete for $\text{COUNT}(A)$ already in the presence of one non-trivial FD, and for the remaining operators in the presence of more than one non-trivial FD.*

For the aggregate operators MIN , MAX , $\text{COUNT}(\ast)$ and SUM and a single FD, the glb- and lub-answers are computed by SQL2 queries (so this is in a sense an analogue of the query rewriting approach for first-order queries discussed earlier). For AVG , however, the polynomial-time algorithm is iterative and cannot be formulated in SQL2.

Example 14. Continuing Example 12, the greatest lower bound answer to the query

```
SELECT MAX(Salary) FROM Emp
```

is computed by the following SQL2 query

```
SELECT MAX(C) FROM
  (SELECT MIN(Salary) AS C
   FROM Emp
   GROUP BY Name) .
```

\square

Arenas et al. [6,8] identify some special properties of conflict graphs in restricted cases, paving the way to more tractable cases. For example, for two FDs and the relation schema in Boyce-Codd Normal Form, the conflict graphs are claw-free and perfect [15], and computing lub-answers to `COUNT(*)` queries can be done in P .

Given the intractability results, it seems appropriate to find approximations to consistent answers to aggregation queries. Unfortunately, “maximal independent set” seems to have bad approximation properties [51].

6 Related work

We only briefly survey related work here. Arenas, Bertossi and Chomicki [4,13] provide a more comprehensive discussion.

There are several similarities between our approach to consistency handling and those followed by the belief revision community [42]. Database repairs (Definition 4) coincide with revised models defined by Winslett [65]. Winslett’s framework is mainly propositional. However, Chou and Winslett [29] study a preliminary extension to first order knowledge bases. Those papers concentrate on the computation of the models of the revised theory, i.e., the repairs in our case. Comparing our framework with that of belief revision, we have an empty domain theory, one model: the database instance, and a revision by a set of ICs. The revision of a database instance by the ICs produces new database instances, the repairs of the original database. We consider a specific notion of minimal change, namely one that minimizes the set of literals in the symmetric differences of two instances. Other possibilities have also been explored in the belief revision community, for example minimizing the cardinality of symmetric difference, as proposed by Dalal [30].

The complexity of belief revision (and the related problem of counterfactual inference which corresponds to our computation of consistent query answers) in the propositional case is exhaustively classified by Eiter and Gottlob [35]. They show, among others, that counterfactual inference under Winslett’s semantics is Π_2^P -complete. Subsequently, they provide a number of restrictions on the knowledge base and the revision formula to reduce the complexity. We note that among the constraint classes considered in the current paper, only universal constraints can be represented propositionally by grounding. However, such grounding results in an unbounded revision formula, which prevents the transfer of any of the polynomial-time upper bounds obtained by Eiter and Gottlob [35] into our framework. Similarly, their lower bounds require different kinds of formulas from those that we use.

The need to accommodate violations of functional dependencies is one of the main motivations for considering disjunctive databases (studied, among others, by Imieliński, van der Meyden, Naqvi, and Vadaparty [53,54,61] and has led to various proposals in the context of data integration (Agarwal et al. [3], Baral et al. [10], Dung [32], and Lin and Mendelzon [58]). There seems to be an intriguing connection between relation repairs w.r.t. FDs and databases with disjunctive

information [61]. For example, the set of repairs of the relation *Person* from Example 3 can be represented as a disjunctive database D consisting of the formulas

$$Person(\text{Brown}, \text{Amherst}, 115 \text{ Klein}) \vee Person(\text{Brown}, \text{Amherst}, 120 \text{ Maple})$$

and

$$Person(\text{Green}, \text{Clarence}, 4000 \text{ Transit}).$$

Each repair corresponds to a minimal model of D and vice versa. We conjecture that the set of all repairs of an instance w.r.t. a set of FDs can be represented as a disjunctive table (with rows that are disjunctions of atoms with the same relation symbol). The relationship in the other direction does not hold, as shown by the following example [8].

Example 15. The set of minimal models of the formula

$$(p(a_1, b_1) \vee p(a_2, b_2)) \wedge p(a_3, b_3)$$

cannot be represented as a set of repairs of any set of FDs. □

Known tractable classes of first-order queries over disjunctive databases typically involve conjunctive queries and databases with restricted OR-objects [53,54]. In some cases, like in Example 3, the set of all repairs can be represented as a table with OR-objects. But in general a correspondence between sets of repairs and tables with OR-objects holds only in the very restricted case when the relation is binary, say $R(A, B)$, and there is one FD $A \rightarrow B$ [8]. Imieliński, van der Meyden, and Vadaparty [54] provide a complete classification of the complexity of conjunctive queries for tables with OR-objects. It is shown how the complexity depends on whether the tables satisfy various schema-level criteria, governing the allowed occurrences of OR-objects. Since there is no exact correspondence between tables with OR-objects and sets of repairs of a given database instance, the results of the paper [54] do not directly translate to our framework, and vice versa. A different interesting direction to explore in this context is to consider *conditional tables*, proposed by Imieliński and Lipski [52], as a representation for infinite sets of repairs, as in Example 8.

There are several proposals for language constructs specifying nondeterministic queries that are related to our approach: *witness*, proposed by Abiteboul, Hull and Vianu [2], and *choice*, proposed by Giannotti, Greco, Pedreschi, Saccà, and Zaniolo [43,44,47]. Essentially, the idea is to construct a maximal subset of a given relation that satisfies a given set of functional dependencies. Since there is usually more than one such subset, the approach yields nondeterministic queries in a natural way. Clearly, maximal consistent subsets (choice models [43]) correspond to repairs. Datalog with choice [43] is, in a sense, more general than our approach, since it combines enforcing functional dependencies with inference using Datalog rules. Answering queries in all choice models ($\forall G$ -queries [47]) corresponds to our notion of computation of consistent query answers (Definition 6). However, the former problem is shown to be *co-NP-complete* and no

tractable cases are identified. One of the sources of complexity in this case is the presence of Datalog rules, absent from our approach. Moreover, the procedure proposed by Greco, Saccà and Zaniolo [47] runs in exponential time if there are exponentially many repairs, as in Example 3. Also, only conjunctions of literals are considered as queries by Greco, Saccà and Zaniolo.

A purely proof-theoretic notion of consistent query answer comes from Bry [17]. This notion, described only in the propositional case, corresponds to evaluating queries after all the tuples involved in inconsistencies have been eliminated. No complexity results have been established for Bry’s approach.

Representing repairs as answer sets of logic programs with disjunction and classical negation has been proposed in a number of papers [5,7,12,34,45,46,49,62]. Those papers consider computing consistent answers to first-order queries. While the approach is very general, no tractable cases beyond those already implicit in the results of Arenas, Bertossi and Chomicki [4] are identified. This is because the classes of logic programs used are Π_2^p -complete [31]. Eiter et al. [34] propose several optimizations that are applicable to logic programming approaches. One is localization of conflict resolution, another - encoding tuple membership in individual repairs using bitvectors, which makes possible efficient computation of consistent query answers using bitwise operators. However, we have seen in Example 3 even in the presence of one functional dependency there may be *exponentially* many repairs [8]. With only 80 tuples involved in conflicts, the number of repairs may exceed 10^{12} ! It is clearly impractical to efficiently manipulate bitvectors of that size.

7 Conclusions and future work

We envision several possible directions for future work.

First, one can consider various *preference orderings* on repairs. Such orderings are often natural and may lead to further tractable cases. Some preliminary work in this direction is reported by Greco et al. [45,48].

Second, the connection between the semantics of repairs and the complexity of repair checking and consistent query answers should be investigated further, in particular the impact of adopting a *cardinality-based* minimality criterion for repairs.

Third, a natural scenario for applying the results developed in this paper is *query rewriting* in the presence of distributed data sources [33,50,57]. Abiteboul and Duschka [1] show, among others, that assuming that data sources are complete has a negative influence on the computational complexity of query answering. However, originally, this line of work didn’t address problems due to database inconsistency. Only recently the research on data integration [14,16,20,56] has started to deal with issues involved in data sources being inconsistent. These works largely remain within the answer-set-based paradigm discussed above. A new scenario for data integration, *data exchange*, has been recently proposed by Fagin et al. [37]. In this scenario, a target database is materialized on the basis of a source database using source-to-target dependencies. In the presence of target

integrity constraints, a suitable consistent target database may not exist. This issue is not considered by Fagin et al. [37]. The work of Cali, Lembo and Rosati [19], discussed earlier, can be viewed as addressing the problem of consistent query answering in a restricted data exchange setting.

Finally, as XML is playing an increased role in data integration, it would be interesting and challenging to develop the appropriate notions of repair and consistent query answer in the context of XML databases. A first attempt in this direction is reported by Flesca et al. [40]. It is limited, however, since it does not consider DTDs. Recent integrity constraint proposals for XML have been made by Buneman et al. [18] and Fan, Kuper, and Siméon [38,39].

Acknowledgments

The comments of Jef Wijsen, Ariel Fuxman, and the anonymous referees are gratefully acknowledged.

References

1. O. Abiteboul and O. Duschka. Complexity of Answering Queries Using Materialized Views. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 254–263, 1998.
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
3. S. Agarwal, A. M. Keller, G. Wiederhold, and K. Saraswat. Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases. In *IEEE International Conference on Data Engineering (ICDE)*, pages 495–504, 1995.
4. M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 68–79, 1999.
5. M. Arenas, L. Bertossi, and J. Chomicki. Specifying and Querying Database Repairs Using Logic Programs with Exceptions. In *International Conference on Flexible Query Answering Systems (FQAS)*, pages 27–41. Springer-Verlag, 2000.
6. M. Arenas, L. Bertossi, and J. Chomicki. Scalar Aggregation in FD-Inconsistent Databases. In *International Conference on Database Theory (ICDT)*, pages 39–53. Springer-Verlag, LNCS 1973, 2001.
7. M. Arenas, L. Bertossi, and J. Chomicki. Answer Sets for Consistent Query Answering in Inconsistent Databases. *Theory and Practice of Logic Programming*, 3(4–5):393–424, 2003.
8. M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science*, 296(3):405–434, 2003.
9. M. Arenas, L. Bertossi, and M. Kifer. Applications of Annotated Predicate Calculus to Querying Inconsistent Databases. In *International Conference on Computational Logic*, pages 926–941. Springer-Verlag, LNCS 1861, 2000.
10. C. Baral, S. Kraus, J. Minker, and V. S. Subrahmanian. Combining Knowledge Bases Consisting of First-Order Theories. *Computational Intelligence*, 8:45–71, 1992.

11. P. Barcelo and L. Bertossi. Repairing Databases with Annotated Predicate Logic. In S. Benferhat and E. Giunchiglia, editors, *Ninth International Workshop on Non-Monotonic Reasoning (NMR02), Special Session: Changing and Integrating Information: From Theory to Practice*, pages 160–170, 2002.
12. P. Barcelo and L. Bertossi. Logic Programs for Querying Inconsistent Databases. In *International Symposium on Practical Aspects of Declarative Languages (PADL)*, pages 208–222. Springer-Verlag, LNCS 2562, 2003.
13. L. Bertossi and J. Chomicki. Query Answering in Inconsistent Databases. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*, pages 43–83. Springer-Verlag, 2003.
14. L. Bertossi, J. Chomicki, A. Cortes, and C. Gutierrez. Consistent Answers from Integrated Data Sources. In *International Conference on Flexible Query Answering Systems (FQAS)*, pages 71–85, Copenhagen, Denmark, October 2002. Springer-Verlag.
15. A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.
16. L. Bravo and L. Bertossi. Logic Programs for Consistently Querying Data Integration Systems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 10–15, 2003.
17. F. Bry. Query Answering in Information Systems with Integrity Constraints. In *IFIP WG 11.5 Working Conference on Integrity and Control in Information Systems*, pages 113–130. Chapman & Hall, 1997.
18. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. *Computer Networks*, 39(5):473–487, 2002.
19. A. Cali, D. Lembo, and R. Rosati. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 260–271, 2003.
20. A. Cali, D. Lembo, and R. Rosati. Query Rewriting and Answering under Constraints in Data Integration Systems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 16–21, 2003.
21. A. Celle and L. Bertossi. Querying Inconsistent Databases: Algorithms and Implementation. In *International Conference on Computational Logic*, pages 942–956. Springer-Verlag, LNCS 1861, 2000.
22. U. S. Chakravarthy, J. Grant, and J. Minker. Logic-Based Approach to Semantic Query Optimization. *ACM Transactions on Database Systems*, 15(2):162–207, 1990.
23. A. Chandra and P. Merlin. Optimal Implementation of Conjunctive Queries in Relational Databases. In *ACM SIGACT Symposium on the Theory of Computing (STOC)*, pages 77–90, 1977.
24. A. K. Chandra and D. Harel. Computable Queries for Relational Databases. *Journal of Computer and System Sciences*, 21:156–178, 1980.
25. J. Chomicki and J. Marcinkowski. On the Computational Complexity of Consistent Query Answers. Technical Report arXiv:cs.DB/0204010, arXiv.org e-Print archive, April 2002.
26. J. Chomicki and J. Marcinkowski. Minimal-Change Integrity Maintenance Using Tuple Deletions. *Information and Computation*, 2004. To appear. Earlier version: Technical Report cs.DB/0212004, arXiv.org e-Print archive.
27. J. Chomicki, J. Marcinkowski, and S. Staworko. Computing Consistent Query Answers Using Conflict Hypergraphs. In *International Conference on Information and Knowledge Management (CIKM)*. ACM Press, 2004. Short version in IIWeb'04.

28. J. Chomicki, J. Marcinkowski, and S. Staworko. Hippo: A System for Computing Consistent Answers to a Class of SQL Queries. In *International Conference on Extending Database Technology (EDBT)*, pages 841–844. Springer-Verlag, LNCS 2992, 2004. System demo.
29. T. Chou and M. Winslett. A Model-Based Belief Revision System. *Journal of Automated Reasoning*, 12:157–208, 1994.
30. M. Dalal. Investigations into a Theory of Knowledge Base Revision. In *National Conference on Artificial Intelligence*, St.Paul, Minnesota, August 1988.
31. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001.
32. Phan Minh Dung. Integrating Data from Possibly Inconsistent Databases. In *International Conference on Cooperative Information Systems (COOPIS)*, pages 58–65, Brussels, Belgium, 1996. IEEE Press.
33. O.M. Duschka, M.R. Genesereth, and A.Y. Levy. Recursive Query Plans for Data Integration. *Journal of Logic Programming*, 43(1):49–73, 2000.
34. T. Eiter, M. Fink, G. Greco, and D. Lembo. Efficient Evaluation of Logic Programs for Querying Data Integration Systems. In *International Conference on Logic Programming (ICLP)*, pages 163–177, 2003.
35. T. Eiter and G. Gottlob. On the Complexity of Propositional Knowledge Base Revision, Updates, and Counterfactuals. *Artificial Intelligence*, 57(2-3):227–270, 1992.
36. S. M. Embury, S. M. Brandt, J. S. Robinson, I. Sutherland, F. A. Bisby, W. A. Gray, A. C. Jones, and R. J. White. Adapting integrity enforcement techniques for data reconciliation. *Information Systems*, 26(8):657–689, 2001.
37. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *International Conference on Database Theory (ICDT)*, pages 207–224. Springer-Verlag, LNCS 2572, 2003.
38. W. Fan, G. Kuper, and J. Simeon. A Unified Constraint Model for XML. *Computer Networks*, 39(5):489–505, 2002.
39. W. Fan and J. Simeon. Integrity Constraints for XML. *Journal of Computer and System Sciences*, 66(1):254–201, 2003.
40. S. Flesca, F. Furfaro, S. Greco, and E. Zumpano. Repairs and Consistent Answers for XML Data with Functional Dependencies. In *International XML Database Symposium*, pages 238–253. Springer-Verlag, LNCS 2824, 2003.
41. A. Fuxman and R. Miller. Towards Inconsistency Management in Data Integration Systems. In *IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, 2003.
42. P. Gärdenfors and H. Rott. Belief Revision. In D. M. Gabbay, J. Hogger, C, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4, pages 35–132. Oxford University Press, 1995.
43. F. Giannotti, S. Greco, D. Saccà, and C. Zaniolo. Programming with Non-determinism in Deductive Databases. *Annals of Mathematics and Artificial Intelligence*, 19(3-4), 1997.
44. F. Giannotti and D. Pedreschi. Datalog with Non-deterministic Choice Computes NDB-PTIME. *Journal of Logic Programming*, 35:75–101, 1998.
45. G. Greco, S. Greco, and E. Zumpano. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *International Conference on Logic Programming (ICLP)*, pages 348–364. Springer-Verlag, LNCS 2237, 2001.

46. G. Greco, S. Greco, and E. Zumpano. A Logical Framework for Querying and Repairing Inconsistent Databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1389–1408, 2003.
47. S. Greco, D. Sacca, and C. Zaniolo. Datalog Queries with Stratified Negation and Choice: from P to D^P . In *International Conference on Database Theory (ICDT)*, pages 82–96. Springer-Verlag, 1995.
48. S. Greco, C. Sirangelo, I. Trubitsyna, and E. Zumpano. Preferred Repairs for Inconsistent Databases. In *International Database Engineering and Applications Symposium (IDEAS)*, pages 202–211. IEEE Computer Society Press, 2003.
49. S. Greco and E. Zumpano. Querying Inconsistent Databases. In *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, pages 308–325. Springer-Verlag, LNCS 1955, 2000.
50. A. Y. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, 10(4):270–294, 2001.
51. D. S. Hochbaum. Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Co., 1997.
52. T. Imieliński and W. Lipski. Incomplete Information in Relational Databases. *Journal of the ACM*, 31(4):761–791, 1984.
53. T. Imieliński, S. Naqvi, and K. Vadaparty. Incomplete Objects - A Data Model for Design and Planning Applications. In *ACM SIGMOD International Conference on Management of Data*, pages 288–297, Denver, Colorado, May 1991.
54. T. Imieliński, R. van der Meyden, and K. Vadaparty. Complexity Tailored Design: A New Design Methodology for Databases With Incomplete Information. *Journal of Computer and System Sciences*, 51(3):405–432, 1995.
55. P. C. Kanellakis. Elements of Relational Database Theory. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 17, pages 1073–1158. Elsevier/MIT Press, 1990.
56. D. Lembo, M. Lenzerini, and R. Rosati. Source Inconsistency and Incompleteness in Data Integration. In *9th International Workshop on Knowledge Representation meets Databases (KRDB'02)*, Toulouse, France, 2002.
57. M. Lenzerini. Data Integration: A Theoretical Perspective. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002. Invited talk.
58. J. Lin and A. O. Mendelzon. Merging Databases under Constraints. *International Journal of Cooperative Information Systems*, 7(1):55–76, 1996.
59. B. Ludäscher, W. May, and G. Lausen. Referential Actions as Logical Rules. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 217–227, 1997.
60. Jim Melton and Alan R. Simon. *SQL:1999 Understanding Relational Language Components*. Morgan Kaufmann, 2002.
61. R. van der Meyden. Logical Approaches to Incomplete Information: A Survey. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 10, pages 307–356. Kluwer Academic Publishers, Boston, 1998.
62. D. Van Nieuwenborgh and D. Vermeir. Preferred Answer Sets for Ordered Logic Programs. In *European Conference on Logics for Artificial Intelligence (JELIA)*, pages 432–443. Springer-Verlag, LNAI 2424, 2002.
63. M. Y. Vardi. The Complexity of Relational Query Languages. In *ACM Symposium on Theory of Computing (STOC)*, pages 137–146, 1982.
64. J. Wijsen. Condensed Representation of Database Repairs for Consistent Query Answering. In *International Conference on Database Theory (ICDT)*, pages 378–393. Springer-Verlag, LNCS 2572, 2003.

65. M. Winslett. Reasoning about Action using a Possible Models Approach. In *National Conference on Artificial Intelligence*, pages 79–83, 1988.