Minimal-Change Integrity Maintenance Using Tuple Deletions ⋆

Jan Chomicki [1]

Department of Computer Science and Engineering, 201 Bell Hall, Univ. at Buffalo, Buffalo, NY 14260-2000. Email: chomicki@cse.buffalo.edu.

Jerzy Marcinkowski

Institute of Informatics, Wroclaw University, Przesmyckiego 20, 51-151 Wroclaw, Poland. Email: 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. We assume that integrity-restoration actions are limited to tuple deletions. We focus on two basic computational issues: *repair checking* (is a database instance a repair of a given database?) and *consistent query answers* [3] (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 cases. We consider denial constraints, general functional and inclusion dependencies, as well as key and foreign key constraints. Our results shed 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

⋆ Research supported by NSF Grant IIS-0119186 and UB faculty start-up funds.

[1] Contact author.

data integration [47]) or is involved in complex, long-running activities like workflows.

How to deal with inconsistent data? The traditional way is not to allow the database to become inconsistent by aborting updates or transactions leading to 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 [48], 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 cause of 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 reported in [49] is an exception).

In our opinion, integrity restoration should be a separate process that is executed after an inconsistency is detected. 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*.

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. If the information is both incorrect and incomplete, then both insertions and deletions should be considered. In this paper we focus on the first case. Since we are working in the context of the relational data model, we consider *tuple deletions*. Such a scenario is common in data warehouse applications where dirty data coming from many sources is cleaned in order to be used as a part of the warehouse itself. On the other hand, in some data integration approaches, e.g., [46, 47], 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 ways to remove integrity violations. We return to the issue of minimal change in Sections 2 and 5.

We claim that a central notion in the context of integrity restoration is that

of a *repair* [3]. A repair is a database instance that satisfies the integrity constraints and *minimally* differs from the original database (which may be inconsistent). Because we consider only *deletions of complete tuples* as ways to restore database consistency, the repairs in our framework are *maximal consistent subsets* of the original database instance.

The basic computational problem in this context is *repair checking*, namely checking whether a given database instance is a repair of the original database. The complexity of this problem is studied in the present paper. Typically, repair checking algorithms can be easily adapted to non-deterministically compute repairs (as we show).

Sometimes when the data is retrieved online from multiple, autonomous sources, it is not possible to restore the consistency by constructing a single repair. In that case one has to settle for computing, in response to queries, *consistent query answers* [3], 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. We note that the notion of consistent query answer proposed in [3] has been used and extended, among others, in [4–8, 10, 11, 14, 17–19, 22, 24, 30, 34, 38, 39, 41, 55, 57]. However, none of these papers presents a comprehensive and complete computational complexity picture. The research on consistent query answers is surveyed in [12] and [23].

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 following parameters:

- the *class of queries*: quantifier-free queries, conjunctive queries, and simple conjunctive queries (conjunctive queries without repeated relation symbols and with limited variable sharing).
- the *class of integrity constraints*: denial constraints, functional dependencies (FDs), inclusion dependencies (INDs), and FDs and INDs together. We also consider practically important subclasses of FDs and INDs: *key* functional dependencies and *foreign key* constraints [27].
- the *number* of integrity constraints.

As a result we obtain several new classes for which both repair checking and consistent query answers are in PTIME:

- queries: quantifier-free, constraints: arbitrary denial;
- queries: simple conjunctive, constraints: functional dependencies (at most one FD per relation);
- queries: quantifier-free or simple conjunctive, constraints: key functional dependencies and foreign key constraints, with at most one key per relation.

Additionally, we show that repair checking (but not consistent query answers) are in PTIME for arbitrary FDs and acyclic INDs. The results obtained are tight in the sense that relaxing any of the above restrictions leads to co-NP-hard problems, as we prove. (This, of course, does not preclude the possibility that introducing *additional*, orthogonal restrictions could lead to more PTIME cases.) To complete the picture, we show that for arbitrary sets of FDs and INDs repair checking is co-NP-complete and consistent query answers is Π_2^p-complete.

Our results shed 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 [49, 53], as ways of performing minimal-change integrity maintenance using tuple deletions.

The plan of the paper is as follows. In Section 2, we define the basic concepts. In Section 3, we consider denial constraints. In Section 4, we discuss INDs together with FDs. In Section 5, we summarize related research and in Section 6 we draw conclusions and discuss future work. An earlier version of the results in Section 3 was presented in [22].

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 $\mathcal{L}$.

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

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

(1) *Denial constraints*: $\mathcal{L}$-sentences

$$\forall \bar{x}_1, \ldots \bar{x}_m.\ \neg[P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m) \wedge \varphi(\bar{x}_1, \ldots, \bar{x}_m)].$$

A denial constraint is *binary* if $m \leq 2$.

(2) *Functional dependencies (FDs)*: $\mathcal{L}$-sentences

$$\forall \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \bar{x}_5.\ \neg[P(\bar{x}_1, \bar{x}_2, \bar{x}_4) \wedge P(\bar{x}_1, \bar{x}_3, \bar{x}_5) \wedge \bar{x}_2 \neq \bar{x}_3],$$

where the $\bar{x}_i$ are sequences of distinct variables. 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 $\bar{x}_3$). Clearly, functional dependencies are a special case of denial constraints.

(3) *Inclusion dependencies (INDs)*: $\mathcal{L}$-sentences

$$\forall \bar{x}_1\ \exists \bar{x}_3.\ [Q(\bar{x}_1) \Rightarrow P(\bar{x}_2, \bar{x}_3)],$$

where the $\bar{x}_i$ are sequences of distinct variables with $\bar{x}_2$ contained in $\bar{x}_1$, and P, Q database relations. Again, this is often written as $Q[Y] \subseteq P[X]$ where X (resp. Y) is the set of attributes of P (resp. Q) corresponding to $\bar{x}_2$. If P and Q are clear from the context, we omit them and write the dependency simply as $Y \subseteq X$. *Full* inclusion dependencies are those expressible without the existential quantifiers.

Several examples of integrity constraints are presented later in Examples 2.1, 2.2, and 2.3.

Given a set of FDs and INDs IC and a relation P with attributes U, a *key* of P is a minimal set of attributes X of P such that IC entails the FD $X \rightarrow U$. In that case, we say that each FD $X \rightarrow Y \in IC$ is a *key* dependency and each IND $Q[Y] \subseteq P[X] \in IC$ is a *foreign key constraint*. If, additionally, X is the primary (one designated) key of P, 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 sys-

tems. The SQL:1999 standard [53] 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 2.1 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.

2.2 Repairs

Given a database instance r, the *set* $\Sigma(r)$ *of facts* of r is the set of ground atomic formulas $\{P(\bar{a}) \mid r \models P(\bar{a})\}$. (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 2.2 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 $\Sigma(r')$ is a maximal subset of $\Sigma(r)$ such that $r' \models IC$.

We denote by $Repairs_{IC}(r)$ the set of repairs of r w.r.t. IC. This set is nonempty, since the empty database instance satisfies every set of denial constraints and INDs.

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* [1,20] are queries of the form

$$\exists \bar{x}_1, \ldots \bar{x}_m.\ [P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m) \wedge \varphi(\bar{x}_1, \ldots, \bar{x}_m)]$$

where the variables of x_i are disjoint from that of x_j if $i \neq j$, and $\varphi(\bar{x}_1, \ldots, \bar{x}_m)$ 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 $c_1(\bar{x}_1) \wedge \cdots \wedge c_m(\bar{x}_m)$.

The following definition is standard:

Definition 2.3 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 result tuples that are unaffected by the violations of IC, even when r violates IC.

Definition 2.4 ([3]) 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 the 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:

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

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

2.5 Examples

Example 2.1 Consider the following instance of a relation *Person*

Name	*City*	*Street*
Brown	Amherst	115 Klein
Brown	Amherst	120 Maple
Green	Clarence	4000 Transit

and the functional dependency *Name* $\rightarrow$ *City 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 query $Person(n, c, s)$ has the tuple (Green,Clarence,4000 Transit) as the only consistent answer. On the other hand, the query $\exists s[Person(n, c, s)]$ has two consistent answers: (Brown,Amherst) and (Green,Clarence). Similarly, the sentence

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

is consistently true. 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 2.2 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:

Employee

SSN	*Name*
123456789	Smith
555555555	Jones
555555555	Smith

Manager

SSN
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 repairs:

Employee

SSN	*Name*
123456789	Smith
555555555	Jones

Manager

SSN
123456789
555555555

and

Employee

SSN	*Name*
555555555	Smith

Manager

SSN
555555555

Example 2.3 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:

$$\forall n, s, m, s', m'.\ \neg[Emp(n, s, m) \wedge Emp(m, s', m') \wedge s > s'].$$

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:

$$\forall n, s, m.\ \neg[Emp(n, s, m) \wedge s > 200000].$$

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 Different notions of repair

The original notion of repair introduced in [3] required that the *symmetric* difference between a database and its repair be minimized. As explained in the introduction, this was based on the assumption that the database may be not only inconsistent but also incomplete [2]. The notion of repair pursued in the current paper (Definition 2.2) reflects the assumption that the database is complete. There are several reasons for this change of perspective. First, for denial constraints integrity violations can only be removed by deleting tuples, so the different notions of repair in fact coincide in this case. Therefore, all the results presented in Section 3 are not affected by the restriction of the repairs to be subsets of the original instance. Insertions can restore integrity only for inclusion dependencies (or, in general for tuple-generating dependencies [1]). Second, even for inclusion dependencies current language standards like SQL:1999 [53] allow only deletions in their repertoire of referential integrity actions. Third, disallowing insertions significantly strengthens the notion of consistent query answer, as demonstrated by the following example.

Example 2.4 Consider a database schema consisting of two relations $P(AB)$ and $S(C)$. The integrity constraints are: the FD $A \rightarrow B$ and the IND $B \subseteq C$. Assume the database instance r_1 consists of $p = \{(a, b), (a, c)\}$ and $s = \{b\}$. Then under Definition 2.2 there is only one repair r_2 consisting of $p' = \{(a, b)\}$ and $s' = s$. On the other hand, under the definition of [3], there is one more repair r_3 consisting of $p'' = \{(a, c)\}$ and $s'' = \{b, c\}$. Therefore, in the first case $P(a, b)$ is consistently true in the original instance r_1, while in the second case it is not. Note that $P(a, c)$ is not consistently true in r_1 either. Thus, in the second case $P(a, b)$ and $P(a, c)$ are treated symmetrically from the point of view of consistent query answering. However, intuitively there is a difference between them. Think of A being the person's name, B her address and S a list of valid addresses. Then only under Definition 2.2 would the single valid address be returned as a consistent answer.

[2] *Incompleteness* here does not mean that the database contains *indefinite information* in the form of nulls or disjunctions [54]. 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 which are, moreover, not very intuitive as ways of fixing an inconsistent database.

Example 2.5 In Example 2.2, allowing insertions gives additionally infinitely many repairs of the form

Employee

SSN	*Name*
123456789	*c*
555555555	Smith

Manager

SSN
123456789
555555555

where c is an arbitrary string different from *Smith*.

While the completeness assumption is natural in the context of data warehousing, there are many applications where a different approach should be taken. For example, in sensor databases the information may be accurate but possibly incomplete. In data integration, the information may be both incorrect and incomplete. For example, if we collect the information about an individual, it may happen that we have no address record for her (incompleteness), or multiple address records for her, coming from different data sources (inconsistency). Different semantics of repair are further discussed in the context of related work in Section 5.

2.7 Computational Problems

We consider here the following complexity classes:

- *PTIME*: 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

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

- *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* [21, 56], since 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.

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 2.6 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

3 Denial constraints

3.1 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 in [5] for FDs only.

Definition 3.1 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

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

such that

$$P_1(\bar{t}_1), P_2(\bar{t}_2), \ldots P_l(\bar{t}_l) \in \Sigma(r),$$

and there is a constraint

$$\forall \bar{x}_1, \bar{x}_2, \ldots \bar{x}_l.\ \neg[P_1(\bar{x}_1) \wedge P_2(\bar{x}_2) \wedge \ldots \wedge P_l(\bar{x}_l) \wedge \varphi(\bar{x}_1, \bar{x}_2, \ldots \bar{x}_l)]$$

in F such that $P_1(\bar{t}_1), P_2(\bar{t}_2), \ldots P_l(\bar{t}_l)$ violate together this constraint, which means that there exists a substitution ρ such that

$$\rho(\bar{x}_1) = \bar{t}_1, \rho(\bar{x}_2) = \bar{t}_2, \ldots \rho(\bar{x}_l) = \bar{t}_l$$

and that $\varphi(\bar{t}_1, \bar{t}_2, \ldots \bar{t}_l)$ 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.

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

Proposition 3.1 *Each repair of r w.r.t. F corresponds to a maximal independent set in $\mathcal{G}_{F,r}$.*

Proposition 3.1 yields the following result:

Proposition 3.2 [7] *For every set of denial constraints F and $\mathcal{L}$-sentence Φ, B_F is in PTIME and $D_{F,\Phi}$ is in co-NP.*

PROOF. Checking whether r' satisfies F is in PTIME. 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.

3.2 Positive results

A set of constraints is *generic* if it does not imply any ground literal. The results in [3] imply the following:

Proposition 3.3 *For every generic set F of binary denial constraints and full inclusion dependencies, and quantifier-free $\mathcal{L}$-sentence*

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

$D_{F,\Phi}$ is in PTIME.

The technique used in [3], query rewriting, does not generalize to non-binary constraints, or queries involving disjunction or quantifiers. However, non-binary constraints and disjunctions do not necessarily lead to intractability, as shown by the following theorem.

Theorem 3.1 *For every set F of denial constraints and quantifier-free $\mathcal{L}$-sentence Φ, $D_{F,\Phi}$ is in PTIME.*

PROOF. We assume the sentence is in CNF [3], i.e., of the form $\Phi = \Phi_1 \wedge \Phi_2 \wedge \ldots \Phi_l$, where each Φ_i is a disjunction of ground literals. Φ is true in every repair of an instance r if and only if each of the clauses Φ_i is true in every repair of r. So it is enough to provide a polynomial algorithm which will check if a given ground clause is consistently *true* in 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

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

where the $\bar{t}_j$'s are tuples of constants. WLOG, we assume that all the facts in the set $\{P_1(\bar{t}_1), \ldots, P_n(\bar{t}_n))\}$ are mutually distinct, and that literals with built-in predicates have been replaced by their truth values and the resulting formula simplified.

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

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

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

[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.

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 PTIME deterministic algorithm.

We prove now the correctness of the above algorithm. First, we show that if the algorithm terminates with YES, then there is a repair r' of r such that

$$r' \models P_1(\bar{t}_1) \wedge P_2(\bar{t}_2) \wedge \ldots \wedge P_m(\bar{t}_m) \wedge \neg P_{m+1}(\bar{t}_{m+1}) \wedge \ldots \wedge \neg P_n(\bar{t}_n).$$

We take r' to be the instance such that $\Sigma(r')$ ($\Sigma(r') \supseteq S$) is the set of facts constructed by the algorithm. Clearly, $r' \models P_1(\bar{t}_1) \wedge P_2(\bar{t}_2) \wedge \ldots \wedge P_m(\bar{t}_m)$. By construction, r' satisfies the integrity constraints and is maximal.

We show now that for every $i = m+1, \ldots, n$, $P_i(\bar{t}_i) \notin \Sigma(r')$. That is clearly the case if $P_i(\bar{t}_i) \notin \Sigma(r)$ because $\Sigma(r') \subseteq \Sigma(r)$. Also, if E_i and E_j, $i, j = m+1, \ldots, n$, are different edges selected by the algorithm, then $P_i(\bar{t}_i) \notin (E_j - \{P_j(\bar{t}_j)\})$. Assume that $P_i(\bar{t}_i) \in (E_j - \{P_j(\bar{t}_j)\})$. Then

$$(E_i - \{P_i(\bar{t}_i)\}) \cup (E_j - \{P_j(\bar{t}_j)\}) \supseteq E_i,$$

which contradicts the definition of the set S, since S cannot contain an edge of the conflict hypergraph. Finally, $P_i(\bar{t}_i)$ cannot have been added in the last stage of the algorithm, since then $\Sigma(r') \supseteq E_i$, which contradicts the fact that r' by construction corresponds to an independent set in the conflict hypergraph.

Second, assume that there is a repair r' of r such that

$$r' \models P_1(\bar{t}_1) \wedge P_2(\bar{t}_2) \wedge \ldots \wedge P_m(\bar{t}_m) \wedge \neg P_{m+1}(\bar{t}_{m+1}) \wedge \ldots \wedge \neg P_n(\bar{t}_n).$$

We show that the algorithm terminates with YES. Note first that for every $P_i(\bar{t}_i) \in \Sigma(r)$ ($i = m+1, \ldots, n$), there is an edge E of the conflict hypergraph such that

$$E - \{P_i(\bar{t}_i)\} \subseteq \Sigma(r').$$

Otherwise, $P_i(\bar{t}_i)$ can be added to $\Sigma(r')$ without violating the integrity constraints. But this would violate the maximality of r'. For every $P_i(\bar{t}_i) \in \Sigma(r)$, denote by E_i some edge such that $E_i - \{P_i(\bar{t}_i)\} \subseteq \Sigma(r')$. We have that

$$\bigcup_{m+1 \leq j \leq n, P_j(\bar{t}_j) \in \Sigma(r)} (E_i - P_i(\bar{t}_i)) \subseteq \Sigma(r').$$

But then there is no edge E such that

$$E \subseteq \bigcup_{m+1 \leq j \leq n, P_j(\bar{t}_j) \in \Sigma(r)} (E_i - P_i(\bar{t}_i)),$$

since otherwise $E \subseteq \Sigma(r')$ which would make r' violate the constraints. We also note that $E_i \neq E_j$ for $i \neq j$, since otherwise $E_i \subseteq \Sigma(r')$. Putting all this together, we can obtain a set of facts S such that

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

and *there is no edge* $E \in \mathcal{G}_{F,r}$ *such that* $E \subseteq S$. ■

The above approach can be generalized to any quantifier-free query, not necessarily ground. The idea is to design a *generator* of ground queries and use the above algorithm as a *checker*. [24] shows how to derive the generator which is a first-order query obtained by transformation from the original one. This shows that computing all consistent answers to quantifier-free queries can be done in polynomial time. [24] also describes a database middleware system for computing consistent query answers, based on this idea.

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 quantifier-free queries but also to all conjunctive queries. As we are going to see now, this is only possibly if the conjunctive queries are suitably restricted.

Theorem 3.2 *Let F be a set of FDs, each dependency over a different relation among $P_1, P_2, \ldots, P_k$. Then for each closed simple conjunctive query Q, there exists a sentence Q' such that for every database instance r, $r \models_F Q$ iff $r \models Q'$. Consequently, $D_{F,Q}$ is in P.*

PROOF. For simplicity, we present the construction assuming that $k = 2$ and each of P_1 and P_2 has three attributes ABC; moving to the general case is straightforward. Assume that an FD $A \rightarrow B$ is defined over both P_1 and P_2. The query Q is of the following form

$$\exists x_1, y_1, z_1, x_2, y_2, z_2.\ [P_1(x_1, y_1, z_1) \wedge P_2(x_2, y_2, z_2) \wedge c_1(x_1, y_1, z_1) \wedge c_2(x_2, y_2, z_2)].$$

Then, the query Q' is as follows:

$$\begin{aligned}
&\exists x_1, y_1, z_1, x_2, y_2, z_2.\ \forall y_1', z_1', y_2', z_2'.\ \exists z_1'', z_2''.\ [P_1(x_1, y_1, z_1) \wedge P_2(x_2, y_2, z_2) \\
&\wedge\ c_1(x_1, y_1, z_1) \wedge c_2(x_2, y_2, z_2) \wedge (P_1(x_1, y_1', z_1') \wedge P_2(x_2, y_2', z_2') \\
&\Rightarrow P_1(x_1, y_1', z_1'') \wedge P_2(x_2, y_2', z_2'') \wedge c_1(x_1, y_1', z_1'') \wedge c_2(x_2, y_2', z_2''))].
\end{aligned}$$

As noted above, in this case the conflict hypergraph is a disjoint union of full multipartite graphs. A repair of the given instance p_1 of P_1 is fixed by choosing for every x a single y such that $P_1(x, y, z)$ is true in this instance for some z; similarly for the given instance p_2 of P_2.

Now if Q' is false, then for each $a_1 \in \pi_A(p_1)$, we can choose $b_1 \in \pi_B(\sigma_{A=a_1}(p_1))$ such that the resulting repair of p_1 falsifies the formula

$$\exists x_1, u_1, z_1.\ P_1(x_1, y_1, z_1) \wedge c_1(x_1, y_1, z_1).$$

Similarly, we can choose a repair of p_2 that falsifies the formula

$$\exists x_2, u_2, z_2.\ P_1(x_2, y_2, z_2) \wedge c_2(x_2, y_2, z_2).$$

Those two repairs taken together constitute a repair falsifying Q. Assume now that Q' is true. Then Q is clearly true in every repair, since for every $a_1 \in \pi_A(p_1)$ and every $b_1 \in \pi_B(\sigma_{A=a_1}(p_1))$, there is a $d_1 \in \pi_C(\sigma_{A=a_1 \wedge B=b_1}(p_1))$ such that $P_1(a_1, b_1, d_1) \wedge c_1(a_1, b_1, d_1)$, and similarly for p_2. Note the importance of the fact that there is no connection, explicit or implied, between the variables occurring in P_1 and c_1 and those in P_2 and c_2.

Q' being a first-order query can be evaluated in PTIME (or more precisely, AC^0) data complexity. Also, note that the size of Q' is linear in the size of Q. ■

What if we consider queries where an existentially-quantified variable in the closed simple conjunctive query Q becomes free? As long as the query does not contain multiple occurrences of the same variable, the set of consistent answers to such a query can obtained by evaluating the transformed query Q' from which the appropriate quantifiers are dropped, thus still in polynomial time. This is because a formula $P_i(\ldots, a, \ldots)$ (where a is a constant and $i = 1, 2$) is equivalent to $\exists x.\ P_i(\ldots, x, \ldots) \wedge x = a$. The consistent answers to a nonground query can be obtained by considering all possible bindings for the variables in the query and evaluating each ground query obtained in this manner. Those bindings are restricted to values coming from the appropriate columns in the database instance. However, the latter restriction is already guaranteed by the form of the query.

We show now that the above results are the strongest possible, since relaxing any of the restrictions leads to co-NP-completeness. This is the case even though we limit ourselves to *key* FDs.

3.3 One key dependency, nonsimple conjunctive query

Theorem 3.3 *There exist a key FD f and a closed conjunctive query*

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

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

PROOF. Reduction from MONOTONE 3-SAT. The FD is $A \rightarrow BC$. Let $\Phi = \phi_1 \wedge \ldots \phi_m \wedge \psi_{m+1} \ldots \wedge \psi_l$ 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 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 Φ if and only if there exists a repair of the database in which Q is false. 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. ■

Remark: The above proof can be easily adapted to the case of a two-literal conjunctive query in which each literal refers to a different binary relation.

3.4 Two key dependencies, single-atom query

By a *bipartite edge-colored graph* we mean a tuple $\mathcal{G} = \langle V, E, B, G\rangle$ such that $\langle V, E\rangle$ is an undirected bipartite graph and $E = B \cup G$ for some given disjoint sets B, G (so we think that each of the edges of $\mathcal{G}$ has one of the two colors).

Definition 3.2 Let $\mathcal{G} = \langle V, E, B, G\rangle$ be a bipartite edge-colored graph, and let $M \subset E$. We say that M is maximal $\mathcal{V}$-free if:

(1) M is a maximal (w.r.t. inclusion) subset of E with the property that:
(*) neither $M(x, y) \wedge M(x, z)$ nor $M(y, x) \wedge M(z, x)$ holds for any x, y, z, (unless $y = z$)
(2) $M \cap B = \emptyset$.

We say that $\mathcal{G}$ has the max-$\mathcal{V}$-free property if there exists M which is maximal $\mathcal{V}$-free.

Of course there always exists a maximal subset M of E satisfying (*): to built such M it is enough to begin with the empty set and keep adding to it edges

from E as long as it is possible without violating (*). But for maximal $\mathcal{V}$-free property it is not enough that such a maximal M exists: the second condition from the definition requires that M should also be disjoint with B. So one can imagine that a bipartite edge-colored graph has the maximal $\mathcal{V}$-free property if a maximal M can be reached by the above construction without adding edges from B.

Lemma 3.1 *Max-$\mathcal{V}$-free is an NP-complete property of bipartite edge-colored graphs.*

PROOF. Reduction from 3-COLORABILITY. Let $\mathcal{H} = \langle U, D\rangle$ be some undirected graph. This is how we define the bipartite edge-colored graph $\mathcal{G}_{\mathcal{H}}$:

(1) $V = \{v_\varepsilon, v'_\varepsilon : v \in U, \varepsilon \in \{m, n, r, g, b\}\}$, which means that there are 10 nodes in the graph $\mathcal{G}$ for each node of $\mathcal{H}$;
(2) $G(v_m, v'_r)$, $G(v_m, v'_b)$, $G(v_n, v'_b)$, $G(v_n, v'_g)$, $G(v_r, v'_m)$, $G(v_b, v'_m)$, $G(v_b, v'_n)$, and $G(v_g, v'_n)$ hold for each $v \in U$;
(3) $B(v_\epsilon, v'_\varepsilon)$ holds for each $v \in U$ and each pair $\epsilon, \varepsilon \in \{r, g, b\}$ such that $\epsilon \neq \varepsilon$;
(4) $B(v_\varepsilon, u'_\varepsilon)$ holds for each $\varepsilon \in \{r, g, b\}$ and each pair $u, v \in U$ such that $D(u, v)$.

Suppose that $\mathcal{H}$ is 3-colorable. We fix a coloring of $\mathcal{H}$ and construct the set M. For each $v \in U$: if the color of v is Red, then the edges $G(v_m, v'_b), G(v_n, v'_g)$ and $G(v_b, v'_m), G(v_g, v'_n)$ are in M. If color of v is Green, then the edges $G(v_m, v'_r), G(v_n, v'_b)$ and $G(v_r, v'_m), G(v_b, v'_n)$ are in M, and if the color of v is Blue, then the edges $G(v_m, v'_r), G(v_n, v'_g)$ and $G(v_r, v'_m), G(v_g, v'_n)$ are in M. It is easy to see that the set M constructed in this way is maximal $\mathcal{V}$-free.

For the other direction, suppose that a maximal $\mathcal{V}$-free set M exists in $\mathcal{G}_{\mathcal{H}}$. Then, for each $v \in U$ there is at least one node among v_r, v_g, v_b which does not belong to any G-edge in M. Let v_ϵ be this node. Also, there is at least one such node (say, v'_ε) among v'_r, v'_g, v'_b. Now, it follows easily from the construction of $\mathcal{G}_{\mathcal{H}}$ that if M is maximal $\mathcal{V}$-free then $\epsilon = \varepsilon$. Let this ϵ be color of v in $\mathcal{G}$. It is easy to check that the coloring defined in this way is a legal 3-coloring of $\mathcal{G}$. ■

Theorem 3.4 *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.*

PROOF. The 2 dependencies are $A \rightarrow BC$ and $B \rightarrow AC$. For a given bipartite edge-colored graph $\mathcal{G} = \langle V, E, B, G\rangle$ we build a database with the tuples (x, y, g) if $G(x, y)$ holds in $\mathcal{G}$ and (x, y, b) if $B(x, y)$ holds in $\mathcal{G}$. Now the

theorem follows from Lemma 3.1 since a repair in which the query Q is not true exists if and only if $\mathcal{G}$ has the max-$\mathcal{V}$-free property. ■

3.5 One denial constraint

By an *edge-colored graph* we mean a tuple $\mathcal{G} = \langle V, E, P, G, B\rangle$ such that $\langle V, E\rangle$ is a (directed) graph and $E = P \cup G \cup B$ for some given pairwise disjoint sets P, G, B (which we interpret as colors). We say that the edge colored graph $\mathcal{G}$ has the $\mathcal{Y}$ property if there are $x, y, z, t \in V$ such that $E(x, y), E(y, z), E(y, t)$ hold and the edges $E(y, z)$ and $E(y, t)$ are of different colors.

Definition 3.3 We say that the edge-colored graph $\langle V, E, P, G, B\rangle$ has the max-$\mathcal{Y}$-free property if there exists a subset M of E such that $P \cap M = \emptyset$ and the following conditions hold:

(1) $\langle V, M, P \cap M, G \cap M, B \cap M\rangle$ does not have the $\mathcal{Y}$-property;
(2) M is a maximal (w.r.t. inclusion) subset of E satisfying the first condition.

Like in the definition of the max-$\mathcal{V}$-free property, also here there always exists $M \subseteq E$ such that conditions 1 and 2 hold. The question is, however, if there exists an M which not only satisfies the conditions but also is disjoint from P.

Lemma 3.2 *Max-$\mathcal{Y}$-free is an NP-complete property of edge-colored graphs.*

PROOF. By a reduction of 3SAT. Let $\Phi = \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_l$ be conjunction of clauses. Let $p_1, p_2, \ldots p_n$ be all the variables in Φ. This is how we define the *edge-colored graph* $\mathcal{G}_\Phi$:

(1) $V = \{a_i, b_i, c_i, d_i : 1 \leq i \leq n\} \cup \{e_i, f_i, g_i : 1 \leq i \leq l\}$, which means that there are 3 nodes in the new graph for each clause in Φ and 4 nodes for each variable.
(2) $P(a_i, b_i)$ and $P(e_j, f_j)$ hold for each suitable i, j;
(3) $G(b_i, d_i)$ and $G(e_j, g_j)$ hold for each suitable i, j;
(4) $B(b_i, c_i)$ holds for each suitable i;
(5) $G(d_i, e_j)$ holds if p_i occurs positively in ϕ_j;
(6) $B(d_i, e_j)$ holds if p_i occurs negatively in ϕ_j;
(7) $E = B \cup G \cup P$.

Now suppose that Φ is satisfiable, and that μ is the satisfying assignment. We define the set $M \subset E$ as follows. We keep in M all the G-edges from item 3 above. If $\mu(p_i) = 1$ then we keep in M all the G-edges leaving d_i (item 5). Otherwise we keep in M all the B-edges leaving d_i (item 6). Obviously,

$M \cap P = \emptyset$. It is also easy to see that M does not have the $\mathcal{Y}$-property and that it is maximal.

In the opposite direction, notice that if an M, as in Definition 3.3 does exist, then it must contain all the G-edges from item 2 above - otherwise a P edge could be added without leading to the $\mathcal{Y}$-property. But this means that, for each i, M can either contain some (or all) of the B-edges leaving d_i or some (or all) of the G-edges. In this sense M defines a valuation of variables. Also, if M is maximal, it must contain, for each j, at least one edge leading to e_j. But this means that the defined valuation satisfies Φ. ■

Theorem 3.5 *There exist a denial constraint f and a closed conjunctive query*

$$Q \equiv \exists x, y.\ [R(x, y, p)],$$

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

PROOF. The denial constraint f is:

$$\forall x, y, z, w, s, s', s''\ \neg[R(x, y, s) \wedge R(y, z, s') \wedge R(y, w, s'') \wedge s' \neq s'']$$

For a given edge-colored graph $\mathcal{G} = \langle V, E, P, G, B\rangle$ we build a database with the tuples $R(x, y, g)$ if $G(x, y)$ holds in $\mathcal{G}$, with $R(x, y, p)$ if $P(x, y)$ holds in $\mathcal{G}$ and with $R(x, y, b)$ if $B(x, y)$ holds in $\mathcal{G}$. Now the theorem follows from Lemma 3.2 since a repair in which the query Q is not true exists iff $\mathcal{G}$ has the max-$\mathcal{Y}$-free property. ■

4 Inclusion dependencies

In Theorem 4.1 below, we formulate a general relationship between the problems of repair checking and consistent query answers in the presence of inclusion dependencies. Intuitively, the complement of the problem of consistent query answers turns out to be at least as hard as the problem of repair checking. Theorem 4.1 enables us to immediately obtain NP-hardness of the problem of consistent query answers from co-NP-hardness of the problem of repair checking.

Theorem 4.1 *If a class of integrity constraints contains inclusion dependencies, then 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 an 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 $(x_1, \ldots, x_k) \in r'$, we add the tuple $(x_1, \ldots, x_k, c_1)$ to s;
- for every tuple $(x_1, \ldots, x_k) \in r - r'$, we add the tuple $(x_1, \ldots, x_k, c_2)$ to s.

Consider also another relation P having a single attribute W , and an inclusion dependency $i_0 : P[W] \subseteq S_0[Z]$. 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. ■

Proposition 4.1 *For every set of INDs I and $\mathcal{L}$-sentence Φ, B_I and $D_{I,\Phi}$ are in PTIME.*

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.

4.1 Single-key relations

We want to identify here the cases where both repair checking and computing consistent query answers can be done in PTIME. 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 PTIME results obtained for FDs in the previous section and in [7].

Lemma 4.1 *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 the 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). ■

We note here that the assumptions of Lemma 4.1 require that each foreign key constraint in I refers to the single key of the referenced relation.

Corollary 4.1 *Under the assumptions of Lemma 4.1, B_{IC} is in PTIME.*

PROOF. Follows from Proposition 3.2. ■

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 PTIME results about consistent query answers obtained for FDs only.

Corollary 4.2 *Let Φ a quantifier-free $\mathcal{L}$-sentence or a simple conjunctive closed $\mathcal{L}$-query. Then under the assumptions of Lemma 4.1, $D_{IC,\Phi}$ is in PTIME.*

PROOF. From Theorem 3.1 and Theorem 3.2. ■

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

4.2 Acyclic inclusion dependencies

For acyclic INDs (and arbitrary FDs), the repair checking problem is still in PTIME. 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 4.1, the problem of consistent query answers becomes intractable, even under acyclicity.

Definition 4.1 [1] 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 4.2 *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 PTIME.*

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 PTIME procedure for computing the repairs w.r.t. $IC = F \cup I$.

To our surprise, Theorem 4.2 is the strongest possible positive result. The problem of consistent query answers is already intractable, even under additional restrictions on the FDs and INDs. To see this let us start by establishing NP-completeness of the *maximal spoiled-free* problem.

By an instance of the maximal spoiled-free problem we mean

$$\mathcal{G} = \langle V, V_1, V_2, V_3, S, E \rangle$$

such that:

(1) $\langle V, E \rangle$ is a ternary undirected hypergraph (so V is a set of vertices and E is a set of *triangles*);
(2) V_1, V_2, V_3 are pairwise disjoint;
(3) $V_1 \cup V_2 \cup V_3 = V$;
(4) Relation E is typed: if $E(a, b, c)$ holds in $\mathcal{G}$ then $a \in V_1$, $b \in V_2$ and $c \in V_3$;
(5) $S \subseteq V_1$ (S will be called set of *spoiled vertices*).

We will consider maximal (with respect to inclusion) sets of disjoint triangles in $\mathcal{G}$. We call a triangle *spoiled* if one of its vertices is spoiled. The *maximal spoiled-free* problem is defined as the problem of deciding, for a given instance $\mathcal{G} = \langle V, V_1, V_2, V_3, S, E \rangle$, if there exists a maximal set $T \subset E$ of disjoint triangles, such that none of the triangles in T is spoiled.

Notice that the general idea here is similar to the one in definitions of maximal-$\mathcal{V}$-free and max-$\mathcal{Y}$-free properties: the question we are considering is not the existence of a set of disjoint triangles, which would be maximal in the class of sets not containing a spoiled triangle: such a set of course always exists. The problem we consider is the existence of a set of disjoint triangles in $\mathcal{G}$ which is not only maximal but also does not contain a spoiled triangle.

As an example, consider $V_1 = \{a_1, a_2, a_3\}$, $V_2 = \{b_1, b_2\}$, $V_3 = \{c_1, c_2\}$, $S = \{a_3\}$ and let E consist of the triangles (a_2, b_1, c_1), (a_2, b_2, c_2), (a_3, b_1, c_1) and (a_3, b_2, c_2). Then any maximal set of disjoint triangles must either contain a spoiled triangle (a_3, b_2, c_2) or a spoiled triangle (a_3, b_1, c_1), and so the instance is not maximal-spoiled-free. If we, however, took the same example with an additional triangle (a_1, b_2, c_2) then the set consisting of (a_1, b_2, c_2) and (a_2, b_1, c_1) would be a maximal set of disjoint triangles without a spoiled triangle, and so the new instance would be maximal-spoiled-free.

Lemma 4.2 *The maximal spoiled-free problem is NP-complete.*

PROOF. By a reduction of 3-colorability. Let $\mathcal{H} = \langle U, D \rangle$ be some undirected graph. We are going to construct the instance of the maximal spoiled-free problem $\mathcal{G}_{\mathcal{H}}$. The construction is a little bit complicated, and we hope to simplify the presentation by the following convention:

Each vertex in V_1 belongs to exactly one triangle in E. So a triangle is fully specified by its vertex in V_2, its vertex in V_3 and by the information if it is spoiled or not.

Now, for each vertex v in U we will have vertices v_r, v_g, v_b, v_p, v_q in V_2 and vertices $v'_r, v'_g, v'_b, v'_p, v'_q$ in V_3. The only nonspoiled triangles will be the defined by the following pairs: $[v_r, v'_p]$, $[v_g, v'_p]$, $[v_g, v'_q]$, $[v_b, v'_q]$, $[v_p, v'_r]$, $[v_p, v'_g]$, $[v_q, v'_g]$, $[v_q, v'_b]$ (so we have 8 nonspoiled triangles for each vertex in U).

There are two kinds of spoiled triangles. For each $v \in U$, and for each pair $\epsilon, \varepsilon \in \{r, g, b\}$ such that $\epsilon \neq \varepsilon$ there is a spoiled triangle $[v_\epsilon, v'_\varepsilon]$ in $\mathcal{G}$. For each $v, u \in U$, such that $D(v, u)$ holds in $\mathcal{H}$, and for each $\epsilon \in \{r, g, b\}$ there is a spoiled triangle $[v_\epsilon, u_\epsilon]$ in $\mathcal{G}$.

Now we need to show that $\mathcal{H}$ is 3-colorable if and only if there exists a maximal set $T \subset E$ of disjoint triangles, such that none of the triangles in T is spoiled.

Let us start from the $\Rightarrow$ direction, which is simple. Consider a coloring of $\mathcal{H}$ with colors r, g and b. Now take T as a set containing, for each vertex v of $\mathcal{H}$ with some color ϵ, all nonspoiled triangles of the form $[v_\alpha, v'_\beta]$ where neither α nor β equals to ϵ. Obviously, T, defined in this way, does not contain spoiled triangles. A simple analysis shows that it is also maximal.

For the other direction suppose that there is a set T of disjoint triangles in $\mathcal{G}$ which is maximal and only contains nonspoiled triangles. It is easy to see that for each v exactly one of the vertices v_r, v_g, v_b is not in any triangle in T, and that also among v'_r, v'_g, v'_b there is exactly one which is not in any triangle in T. If they were different, in the sense that first of them were v_ε and the second v'_ϵ, for $\epsilon \neq \varepsilon$, then a spoiled triangle $[v_\varepsilon, v'_\epsilon]$ could be added to T what contradicts its maximality. So they are equal, and in a natural way they define a color of v. Now we need to prove that the coloring of $\mathcal{H}$ defined in this way is a legal one. But if $D(u, v)$ holds in $\mathcal{H}$ then there is spoiled triangle $[v_\epsilon, u_\epsilon]$ in $\mathcal{G}$ for each $\epsilon \in \{r, g, b\}$. So if the colors of v and u were both equal to some ϵ, then we could add this spoiled triangle, and T would not be maximal. ■

Theorem 4.3 *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.*

PROOF. The database schema consists of a unary relation P, a binary relation $Q(Q_1, Q_2)$ and a ternary relation $R(R_1, R_2, R_3)$. The columns Q_1,R_1,R_2,R_3 are keys, with Q_1 and R_1 being the primary keys. The foreign key dependencies are $P \subseteq Q_1$ and $Q_2 \subseteq R_1$. For a given instance $\mathcal{G}$ of the maximal spoiled-free problem we will construct a database instance r, and a query Φ such that $\mathcal{G}$ has the maximal spoiled-free property if and only if there is a repair r' of r with respect to IC such that Φ is not true in r'.

We define the relation P as a single fact $P(a)$. The relation Q is defined as a set of facts $\{Q(a, s) : s \in S\}$, where S is the set of spoiled vertices from $\mathcal{G}$. Finally, R is the hypergraph from $\mathcal{G}$. The query Φ is $P(a)$.

The repairs of R with respect to the key dependencies correspond to maximal sets of disjoint triangles in $\mathcal{G}$. If $\mathcal{G}$ has the maximal spoiled-free property then there exists a repair of R which does not contain any tuple of the form $R(s, u, v)$ with $s \in S$. But then the only way to repair Q is it take the empty relation, and, consequently, the only way to repair P is to take the empty relation. So if $\mathcal{G}$ has the maximal spoiled-free property then Φ indeed is not true in all repairs. For the other direction notice that if if each repair of R it is a tuple of the form $R(s, u, v)$ with $s \in S$ then each repair of Q is nonempty and in consequence each repair of P consists of the single atom $P(a)$, so then Φ is indeed true in all repairs. ■

Notice that it follows from the proof that in presence of inclusion dependencies deciding if a ground query (in this case $P(a)$) is consistently true in a database is as hard as deciding if a query of the form $\exists s\, Q(a, s)$ is consistently true.

4.3 Relaxing acyclicity

We show here that relaxing the acyclicity assumption in Theorem 4.2 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.

4.3.1 One FD, one IND

Theorem 4.4 *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.*

PROOF. We will check here whether the empty set is a repair. The database schema consists of one relation $R(A_1, A_2, A_3, A_4)$ and the constraints in IC are $A_1 \rightarrow A_2$ and $A_3 \subseteq A_4$.

Consider a propositional formula $\Phi = \phi_1 \wedge \phi_2 \wedge \ldots \phi_m$, where ϕ_i are clauses. Let r_Φ consist of the facts $R(p_j, 0, \phi_i, \phi_{i+1})$ such that p_j occurs negatively in ϕ_i and of the facts $R(p_j, 1, \phi_i, \phi_{i+1})$ such that p_j occurs positively in ϕ_i where the addition $i + 1$ is meant modulo the number m of clauses in Φ. We want to show that $\emptyset$ is a repair of r_Φ with respect to IC if and only if Φ is not satisfiable.

For the *only if* direction notice that if ρ is a satisfying assignment of Φ then the subset of r_Φ consisting of all the facts of the form $R(p, \rho(p), \phi_i, \phi_{i+1})$ is a repair, and obviously $\emptyset$ is not a repair then.

For the opposite direction first notice that a repair r' of r_Φ which is nonempty contains some fact of the form $R(\_, \_, \phi_i, \phi_{i+1})$. So, by inclusion $A_3 \subseteq A_4$ it must also contain some fact of the form $R(\_, \_, \phi_{i-1}, \phi_i)$. By induction we show that

(*) for every clause ϕ_j from Φ there is a fact of the form $R(\_, \_, \phi_j, \phi_{j+1})$ in r'.

Now we make use of the functional dependency $A_1 \rightarrow A_2$. If r' is a repair of r_Φ then for each variable p there are either only facts of the form $R(p, 0, \_, \_)$ in r' or only facts of the form $R(p, 1, \_, \_)$. Define the assignment $\rho(p)$ as 1 if there is some fact of the form $R(p, 1, \_, \_)$ in r' and as 0 otherwise. It follows from the construction of r_Φ that if a clause of the form $R(\_, \_, \phi_j, \phi_{j+1})$ is in r' then ρ satisfies ϕ_j. Together with (*) this completes the proof. ■

4.3.2 Key FDs and foreign key constraints

Theorem 4.5 *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.*

PROOF. Again we consider checking whether the empty set is a repair. The schema consists of 10 binary relations: $R(A, B), R_{i,j}(A_{i,j}, B_{i,j})$ with $1 \leq i, j \leq 3$. For each pair (i, j) both the key dependencies $A_{i,j} \rightarrow B_{i,j}$ and $B_{i,j} \rightarrow A_{i,j}$ are in IC, with $A_{i,j}$ as the primary key of the respective relation. The relation R is constrained by a single key dependency $B \rightarrow A$. The inclusion constraints are $B_{i,j} \subseteq B$, for each pair i, j and $A \subseteq A_{i,j}$, also for each pair i, j.

Consider a propositional formula $\Phi = \phi_1 \wedge \phi_2 \wedge \ldots \phi_m$, where ϕ_i are clauses. We assume that none of the clauses in Φ contains more than 3 literals, that each variable occurs at most 3 times in Φ, and that the number of variables in Φ is equal to the number m of clauses in the formula. It is easy to prove that satisfiability is NP-complete even for formulae of this kind. For the formula

Φ we built a database instance r_Φ: in the the relation R we remember the formula Φ: it consists of such pairs (w, ϕ) that w is a literal, ϕ is a clause from Φ and w occurs in ϕ. The definitions of the relations $R_{i,j}$ are a little bit more complicated. The relation $R_{i,j}$ consists of $2m$ tuples $(p_l, \phi_{s(i,j,l)})$, and $(\neg p_l, \phi_{s(i,j,l)})$, with s still to be defined. The function s will be from $\{1,2,3\} \times \{1,2,3\} \times \{1,2,\ldots m\}$ to $\{1,2,\ldots m\}$ and, more precisely, it is going to be a permutation of $\{1,2,\ldots m\}$ for every fixed pair (i,j). Define $s(i,j,l)$ as n if p_l (or $\neg p_l$) occurs in the clause ϕ_{n+1} (where addition is modulo the number of clauses m), if p_l is the ith variable in this clause, and if it is jth occurrence of p_l in Φ. Now, for each (i,j) let $s(i,j,\_)$ be any permutation consistent with the above definition. It follows directly from our construction that:

Lemma 4.3 *For each clause ϕ_n from Φ and for each variable p occurring in ϕ_n there is a relation $R_{i,j}$ such that the tuples (p, ϕ_{n-1}) and $(\neg p, \phi_{n-1})$ are in $R_{i,j}$.*

We want to show that $\emptyset$ is a repair of r_Φ with respect to IC if and only if Φ is not satisfiable.

The *only if* direction is simple. Assume that Φ is satisfiable and let ρ be a satisfying assignment. In each tuple in each of the relations $R, R_{i,j}$ in r_Φ the first argument is always a literal. Let r' be a subset of r_Φ consisting of such facts $R(w,\phi)$ or $R_{i,j}(w,\phi)$ that $\rho(w) = 1$. The key constraints for $R_{i,j}$ are satisfied in r'. The inclusion constraints $B_{i,j} \subseteq B$ are satisfied because, since ρ was an assignment satisfying Φ, $B = \{\phi_1, \phi_2, \ldots \phi_m\}$ (abusing the notation a little bit we identify a column of a relation with the set of values appearing in this column). Also the inclusions $A \subseteq A_{i,j}$ hold. But the key dependency $B \to A$ does not need to hold in r' (this is because there is possibly more than one literal w in some clause such that $\rho(w) = 1$). To construct a nonempty repair of r_Φ take now r'' built with the same relations $R_{i,j}$ as r' and with relation R being the result of selecting from the relation R in r' exactly one tuple (w,ϕ) for each ϕ.

The *if* direction is more complicated. Consider a repair r' of r_Φ. Since $B_{i,j} \to A_{i,j}$, in each of the relations $R_{i,j}$ (which here, and till the end of this proof, means the relation in r'), for each clause $\phi_s(i,j,l)$ at most one of the tuples $(p_l, \phi_{s(i,j,l)})$ and $(\neg p_l, \phi_{s(i,j,l)})$ can be in $R_{i,j}$. This implies that $A_{i,j}$ is a consistent set of literals: for each l at most one of the literals $p_l, \neg p_l$ can be in $A_{i,j}$. But $A \subseteq A_{i,j}$ so A also is a consistent set. Φ is not satisfiable, which exactly means that for every consistent set of literals there is a clause in Φ which is disjoint with this set. So let ϕ_l be such a clause in Φ that none of the literals from ϕ_l is in A. This means that ϕ_l is not in B (do not forget we mean the relations A and B in r' here) . Consider the clause ϕ_{l+1}. By Lemma 4.3 for each variable p from ϕ_{l+1} there is a relation $R_{i,j}$ such that the tuples (p, ϕ_l) and $(\neg p, \phi_l)$ are in $R_{i,j}$ in r_Φ. But, by the inclusion constraints, each of the $B_{i,j}$ should be a

subset of B, so since ϕ_l is not in B in r' it is also not in any of the $B_{i,j}$ in r'. While removing ϕ_l from $B_{i,j}$ we also delete the variables occurring in a tuple of $R_{i,j}$ together with ϕ_l. This means that for each variable p from the clause ϕ_{l+1} there is a relation $R_{i,j}$ such that neither p nor $\neg p$ is in $A_{i,j}$. But A is a subset of each of the $A_{i,j}$. This means that none of the literals from ϕ_{l+1} can be in A. So ϕ_{l+1} cannot be in B! Now, using this argument m times we can remove all the tuples from the relations, thus proving that r' is empty. ■

4.4 Arbitrary FDs and INDs

Theorem 4.6 *The repair checking problem for arbitrary FDs and INDs is co-NP-complete.*

PROOF. Co-NP-hardness was established earlier in this section. The membership in co-NP follows from the definition of repair: to prove that r' is not a repair of r it is enough to guess a consistent r'' such that $\Sigma(r') \subset \Sigma(r'') \subseteq \Sigma(r)$. ■

Theorem 4.7 *The consistent query answers problem for arbitrary FDs and INDs is Π_2^p-complete.*

PROOF. The membership in Π_2^p follows from the definition of consistent query answer: query is not consistently true if it is false in some repair, and checking if a given set is a repair is in co-NP. We show Π_2^p-hardness below.

Consider a quantified boolean formula β of the form

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

where ψ is quantifier-free and equals to $\psi_1 \wedge \psi_2 \wedge \ldots \psi_m$, 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 occurence 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 $p_1, p_2, \ldots p_k$ that the formula $\sigma(\beta)$ (with free variables $q_1, q_2, \ldots q_l$) 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 $p_1, p_2, \ldots p_k$. Since β is true, there exists a valuation σ' of the variables $q_1, q_2, \ldots q_l$ 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. ■

5 Related work

A comprehensive discussion of related research up to 2002 can be found in [12]. We briefly survey that research here. We also address in detail the issues arising in more recent work, namely [17] and [57]. Another survey, focusing on computational complexity issues, is [23].

5.1 Repairs and consistent query answers

A purely proof-theoretic notion of consistent query answer comes from Bry [15]. This notion, described only in the propositional case, corresponds to evaluating queries after all the tuples involved in inconsistencies have been eliminated. There is no model-theoretic semantics and the notion of minimal change is not captured.

The paper [3] introduced the notions of repair and consistent query answer used by most of the subsequent research. A repair in this framework is a consistent instance minimally different (in the sense of symmetric difference of sets of facts) from the original database instance. For denial constraints, that notion coincides with the notion of repair used in the present paper. The paper [3] proposed computing consistent query answers through *query rewriting*. The approach of [3], further refined in [19], is limited to queries that are conjunctions of literals and binary universal integrity constraints. The papers [5, 7] characterized the complexity of computing consistent query answers in the context of FDs and scalar aggregation queries.

Representing repairs as answer sets of logic programs with disjunction and classical negation has been proposed in [4, 6, 11, 30, 38, 39, 41, 55]. 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 [3] are identified. This is because the classes of logic programs used are Π_2^p-complete [26]. The paper [30] proposes 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 2.6 even in the presence of one functional dependency there may be *exponentially* many repairs [7]. 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.

In [17] the complexity of query answering is considered, when the database is possibly not only inconsistent but also incomplete. Like in [3] and the follow-up work, consistency is defined by means of integrity constraints, and an answer to a query is understood to be true if it is true in all possible repairs. Six definitions of the notion of repair are considered in [17]. Each of them 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 be a superset of the database; the exact semantics – that it be equal to the database; and the complete semantics – that it be 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 in [17].

To present the decidable cases identified in [17], we need to introduce some definitions.

Definition 5.1 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 5.1 [17] *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 $D_{IC,Q}$ is in PTIME (under sound semantics).*

Notice that Theorem 5.1 to some degree parallels Lemma 4.1. 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 the notion proposed in [3]. Finally, loosely-sound semantics requires only that the set of deleted tuples be minimized.

[17] shows 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 5.2 [17] *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 $D_{IC,Q}$ is co-NP-complete (under loosely-sound semantics) and Π_2^p-complete (under loosely-exact semantics).*

Contrasting Theorem 5.2 with Theorem 5.1, 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 layer of nondeterminism.

We conclude by noting that none of the six notions of repair from [17] coincides with that of the present paper. The latter notion, by forcing the repairs to be subsets of the original database, makes the problem of consistent query answers decidable.

5.3 Repairing by attribute modification

In [3] and the follow-up papers, 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 in [57] 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. The motivating example in [57] 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.

The basic idea of the approach proposed in [57] is to define the notion of repair by means of the ordering defined by the *subsumption of tableaux*, instead of the ordering defined by the inclusion of relations. A tableau is a generalization of a relation: tuples can have not only constants but also variables as components. [57] considers only single-relation databases.

Definition 5.2 [57] 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 5.3 [57] 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 Wijsen's framework is that of Definition 2.4 in which the notion of repair of Definition 2.2 is substituted by that of Definition 5.3.

Example 5.1 If the dioxin database contains just one tuple

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

then there are two fixes of it:

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

and

$$\langle 120, 17Jan2002, y, ICI, 150\rangle.$$

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 in [57] consists of tuple- and equality-generating dependencies [1, 45]. The first are simply universal Horn constraints, the second – restricted denial constraints. The queries in [57] are conjunctive queries.

[57] considers cases when 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, NP-hardness of fix checking has been established [58]:

Theorem 5.3 *There exists a denial constraint ϕ_0 with one database literal, such that repair checking is NP-hard.*

It is likely that a similar result holds for consistent query answers. Note that for denial constraints repair checking under Definition 2.2 is in PTIME. Thus, unless $PTIME = NP$, there is a considerable computational price for using the framework of [57]. However, [58] shows that if fixes are appropriately restricted, fix checking becomes a PTIME problem.

5.4 Belief revision

There are several similarities between our approach to consistency handling and those followed by the belief revision/update community [35]. Database repairs (Definition 2.2) coincide with revised models defined by Winslett in [59]. The treatment in [59] is mainly propositional, but a preliminary extension to first order knowledge bases can be found in [25]. 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. 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 was exhaustively classified by Eiter and Gottlob [31]. Among the constraint classes considered in the current paper, only denial constraints can be represented propositionally by grounding. However, such grounding results in an unbounded update formula, which prevents the transfer of any of the PTIME upper bounds from [31] 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 [43, 54] and has led to various proposals in the context of data integration [2, 9, 28, 48]. There seems to be an intriguing connection between relation repairs w.r.t. FDs and databases with disjunctive information [54]. For example, the set of repairs of the relation *Person* from Example 2.6 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 folowing example [7].

Example 5.2 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 [43, 44]. In some cases, like in Example 2.6, the set of all repairs can be represented as a table with OR-objects. But in general this is not the case [7].

Example 5.3 Consider the following set of FDs $F = \{A \rightarrow B, A \rightarrow C\}$, which is in BCNF. The set of all repairs of the instance $\{(a_1, b_1, c_1), (a_1, b_2, c_2)\}$ cannot be represented as a table with OR-objects. □

The relationship in the other direction, from tables with OR-objects to sets of repairs, also does not hold.

Example 5.4 Consider the following table with OR-objects:

OR(a,b)	c
a	OR(c,d)

It does not represent the set of all repairs of any instance under any set of FDs. □

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 $A \rightarrow B$. The paper [44] contains 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 [44] do not directly translate to our framework, and vice versa.

There are several proposals for language constructs specifying nondeterministic queries that are related to our approach (*witness* [1], *choice* [36, 37, 40]). 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 [36]) correspond to repairs. Datalog with choice [36] 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 [40]) corresponds to our notion of computation of consistent query answers (Definition 2.4). 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 in [40] runs in exponential time if there are exponentially many repairs, as in Example 2.6. Also, only conjunctions of literals are considered as queries in [40].

5.6 Functional and inclusion dependencies

It is interesting to contrast our results in Section 4 with the classical results about the implication problem for FDs and INDs [1]. This problem is undecidable in general but becomes decidable under suitable restrictions on INDs. For instance, it is decidable in PTIME if the INDs are unary and in EXPTIME if the INDs are acyclic. The problems discussed in our paper are all in Π_2^p (Section 4). The role the syntactic restrictions play in this context is different. The restriction to unary INDs is not helpful because such INDs already suffice to establish the lower bounds, c.f., Theorem 4.7. The restriction to acyclic INDs makes the repair checking problem tractable (Theorem 4.2) but not so the problem of consistent query answers (Theorem 4.3).

In [50], several classes of FDs and INDs were identified for which the implication problem does not exhibit any interaction between the FDs and the INDs. I.e., a set of constraints implies an FD (resp. an IND) iff the FDs (resp. the INDs) in this set imply it. Unfortunately, the syntactic restrictions on constraints that guarantee no interaction in the above sense do not play a similar role in our context. It seems that the notion of maximality present in the repair definition forces a relationship between the FDs and the INDs that is much tighter than the one implicit in the implication problem.

In [50, 51], it is investigated what kind of relational schemas and integrity constraints can result from mapping an Entity-Relationship schema (this is a common way of designing relational schemas). Acyclicity of INDs is a necessary requirement, thus repair checking is tractable in this case. However, it turns out that the schema from Theorem 4.3 could result from such a mapping. Thus, even restricting the relational schemas to those that correspond to Entity-Relationship schemas does not guarantee the tractability of consistent query answers.

The semantics of referential integrity actions are captured using stable models of logic programs with negation in [52].

6 Conclusions and future work

In this paper we have investigated the computational complexity issues involved in minimal-change integrity maintenance using tuple deletions, in the presence of denial constraints and inclusion dependencies. We have identified several tractable cases and shown that generalizing them leads to intractability.

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 in [38]. Second, a natural scenario for applying the results developed in this paper is *query rewriting* in the presence of distributed data sources [29, 42, 47]. Recent work in this area has started to address the issues involved in data sources being inconsistent [13, 14, 18, 46]. 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. Recent integrity constraint proposals for XML include [16, 32, 33].

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] C. Baral, S. Kraus, J. Minker, and V. S. Subrahmanian. Combining Knowledge Bases Consisting of First-Order Theories. *Computational Intelligence*, 8:45–71, 1992.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. *Computer Networks*, 39(5):473–487, 2002.

[17] A. Calì, 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.

[18] A. Calì, 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.

[19] 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.

[20] 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.

[21] A. K. Chandra and D. Harel. Computable Queries for Relational Databases. *Journal of Computer and System Sciences*, 21:156–178, 1980.

[22] 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.

[23] J. Chomicki and J. Marcinkowski. On the Computational Complexity of Minimal-Change Integrity Maintenance in Relational Databases. In L. Bertossi, A. Hunter, and T. Schaub, editors, *Inconsistency Tolerance*. Springer-Verlag, 2004. To appear.

[24] 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.

[25] T. Chou and M. Winslett. A Model-Based Belief Revision System. *Journal of Automated Reasoning*, 12:157–208, 1994.

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

[27] C. J. Date. Referential Integrity. In *International Conference on Very Large Data Bases (VLDB)*, pages 2–12, 1981.

[28] 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.

[29] 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.

[30] 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.

[31] T. Eiter and G. Gottlob. On the Complexity of Propositional Knowledge Base Revision, Updates, and Counterfactuals. *Artificial Intelligence*, 57(2-3):227–270, 1992.

[32] W. Fan, G. Kuper, and J. Simeon. A Unified Constraint Model for XML. *Computer Networks*, 39(5):489–505, 2002.

[33] W. Fan and J. Simeon. Integrity Constraints for XML. *Journal of Computer and System Sciences*, 66(1):254–201, 2003.

[34] 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.

[35] 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.

[36] 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.

[37] F. Giannotti and D. Pedreschi. Datalog with Non-deterministic Choice Computes NDB-PTIME. *Journal of Logic Programming*, 35:75–101, 1998.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] A. Y. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, 10(4):270–294, 2001.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] M. Lenzerini. Data Integration: A Theoretical Perspective. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002. Invited talk.

[48] J. Lin and A. O. Mendelzon. Merging Databases under Constraints. *International Journal of Cooperative Information Systems*, 7(1):55–76, 1996.

[49] 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.

[50] H. Mannila and K-J. Räihä. *The Design of Relational Databases*. Addison-Wesley, 1992.

[51] V. M. Markowitz and J.A. Makowsky. Identifying Extended Entity-Relationship Object Structures in Relational Schemas. *IEEE Transactions on Software Engineering*, 16(8):777–790, 1990.

[52] Wolfgang May and Bertram Ludäscher. Understanding the global semantics of referential actions using logic rules. *ACM Transactions on Database Systems*, 27(4):343–397, 2002.

[53] Jim Melton and Alan R. Simon. *SQL:1999 Understanding Relational Language Components*. Morgan Kaufmann, 2002.

[54] 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.

[55] 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.

[56] M. Y. Vardi. The Complexity of Relational Query Languages. In *ACM Symposium on Theory of Computing (STOC)*, pages 137–146, 1982.

[57] 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.

[58] J. Wijsen. Making More Out of an Inconsistent Database. In *East-European Conference on Advances in Databases and Information Systems (ADBIS)*. Springer, 2004.

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