Consistent Query Answering (CQA)

Jef Wijsen

University of Mons, Belgium

EDBT-Intended Summer School 2022

Slides for Download

Table of Contents

Table of Contents

Accompanying Paper

SIGMOD RECORD Web Edition

Current Issue Previous Issues Info for Authors Record

Database Principles

Foundations of Query Answering on Inconsistent Databases

Jef Wijsen

Available in: PDF

Published in September 2019 (Vol.48 No.3)

CQA Started at ACM PODS 1999 [ABC03]

Consistent Query Answers in Inconsistent Databases

Marcelo Arenas
Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de Computación
Casilla 306, Santiago 22, Chile
marenas@ing.puc.cl

Leopoldo Bertossi
Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de Computación
Casilla 306, Santiago 22, Chile
bertossi@ing.puc.cl

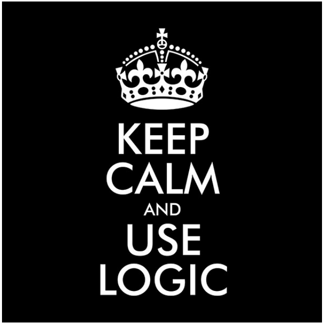
Jan Chomicki
Monmouth University
Department of Computer Science
West Long Branch, NJ 07764
chomicki@monmouth.edu

Abstract

In this paper we consider the problem of the logical characterization of the notion of consistent answer in a relational database that may violate given integrity constraints.

How to Deal with Inconsistent Data?

How to Deal with Inconsistent Data?



Example (Database repairs and Consistent Query Answers (CQA))

$$\Sigma = \{\ \mathit{TAUGHT\text{-}BY} : \{\mathit{Course}\} \rightarrow \{\mathit{Teacher}, \mathit{Semester}\}\ \}$$

TAUGHT-BY

Course	*Teacher*	*Semester*
CS402	D. Maier	Fall
CS402	J. Ullman	Fall

Consistency can be restored by deleting either the first or the last tuple.
We thus find two repairs, both of which are equally "good."

How shall we answer queries?
CQA: *"Intersect query answers on all repairs."*

Courses taught in the Fall semester?
Consistent answer is {CS402}.

Courses taught by D. Maier?
Consistent answer is $\emptyset$.

Example (Database repairs and Consistent Query Answers (CQA))

$$\Sigma = \{\ \textit{TAUGHT-BY} : \{\textit{Course}\} \rightarrow \{\textit{Teacher}, \textit{Semester}\}\ \}$$

TAUGHT-BY

Course	*Teacher*	*Semester*
CS402	D. Maier	Fall
CS402	J. Ullman	Fall

Consistency can be restored by deleting either the first or the last tuple. We thus find two repairs, both of which are equally "good."

How shall we answer queries?
CQA: *"Intersect query answers on all repairs."*

Courses taught in the Fall semester?
Consistent answer is {CS402}.

Courses taught by D. Maier?
Consistent answer is $\emptyset$.

Example (Database repairs and Consistent Query Answers (CQA))

$$\Sigma = \{\ \mathit{TAUGHT\text{-}BY} : \{\mathit{Course}\} \rightarrow \{\mathit{Teacher}, \mathit{Semester}\}\ \}$$

TAUGHT-BY

Course	*Teacher*	*Semester*
CS402	D. Maier	Fall
CS402	J. Ullman	Fall

Consistency can be restored by deleting either the first or the last tuple. We thus find two repairs, both of which are equally "good."

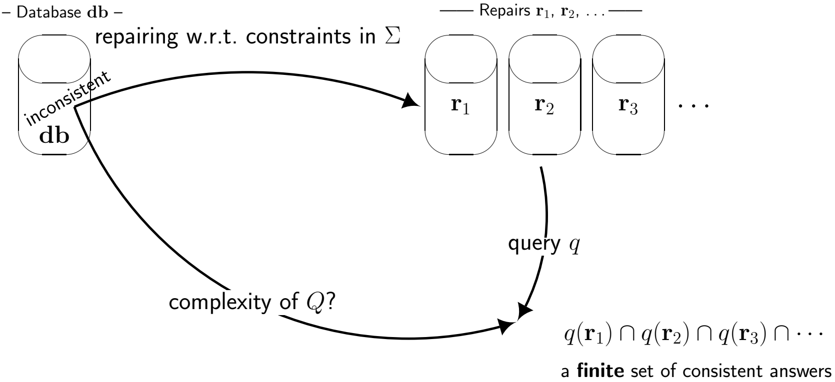
How shall we answer queries?
CQA: *"Intersect query answers on all repairs."*

Courses taught in the Fall semester?
Consistent answer is {CS402}.

Courses taught by D. Maier?
Consistent answer is $\emptyset$.

Consistent Query Answering

- How to answer a query q on a database **db** that is inconsistent w.r.t. a set Σ of constraints?



Example (CQA)

$$\Sigma = \{\ TAUGHT\text{-}BY : \{Course\} \rightarrow \{Teacher, Semester\}\ \}$$

TAUGHT-BY

Course	*Teacher*	*Semester*
CS402	D. Maier	Fall
CS402	J. Ullman	Fall

Courses taught by D. Maier?

$$q = \{x \mid \exists z\ TAUGHT\text{-}BY(x, \text{'D. Maier'}, z)\}$$

$$Q = \{x \mid \exists z\ TAUGHT\text{-}BY(x, \text{'D. Maier'}, z) \wedge \neg\exists y \exists z\, (TAUGHT\text{-}BY(x, y, z) \wedge (y \neq \text{'D. Maier'}))\}$$

Example (CQA)

$$\Sigma = \{\ \mathit{TAUGHT\text{-}BY} : \{\mathit{Course}\} \rightarrow \{\mathit{Teacher}, \mathit{Semester}\}\ \}$$

TAUGHT-BY

Course	*Teacher*	*Semester*
CS402	D. Maier	Fall
CS402	J. Ullman	Fall

Courses taught by D. Maier?

$$q = \{x \mid \exists z\ \mathit{TAUGHT\text{-}BY}(x, \text{‘D. Maier’}, z)\}$$

$$Q = \{x \mid \exists z\ \mathit{TAUGHT\text{-}BY}(x, \text{‘D. Maier’}, z) \wedge \neg\exists y \exists z\,(\mathit{TAUGHT\text{-}BY}(x, y, z) \wedge (y \neq \text{‘D. Maier’}))\}$$

CQA versus Data Cleaning

- Data cleaning is often a long and expensive process. Queries are subsequently posed against the cleaned database.
- CQA returns sound (but maybe incomplete) query answers without the need for actual cleaning or repairing.

Database Instances

The following definitions are relative to:

- a relational schema (= finite set of relation names with associated arities), and
- a finite set Σ of integrity constraints (= first-order logic sentences, mostly of syntactically restricted forms, called dependencies).

A database instance **db** (or simply database) interprets every k-ary relation name R in the schema by a **finite** k-ary relation, denoted $R^{\mathbf{db}}$.

If $\vec{a} \in R^{\mathbf{db}}$, we also say $R(\vec{a}) \in \mathbf{db}$, considering a database as a set of facts.

Under the **named perspective**, every relation name R is associated with a finite (and linearly ordered) set $sort(R)$ of attributes.

Database Repairs

A database **db** is consistent if **db** $\models \Sigma$.

A repair of a (possibly inconsistent) database **db** is a consistent database that differs from **db** in a minimal way (which will be defined later).

Intuitively, we can view each repair as a **possible world**, which brings us in the realm of **incomplete databases** [IJ84], [AHV95, Chapter 19].

Given a database **db**, the consistent answer to a first-order query $q(x_1, \ldots, x_n)$ is the set of tuples $(a_1, \ldots, a_n)$ such that every repair of **db** satisfies $q(a_1, \ldots, a_n)$.

From an incomplete database viewpoint, consistent answers are **certain answers**.

Database Repairs

A database **db** is consistent if **db** $\models \Sigma$.

A repair of a (possibly inconsistent) database **db** is a consistent database that differs from **db** in a minimal way (which will be defined later).

Intuitively, we can view each repair as a **possible world**, which brings us in the realm of **incomplete databases** [IJ84], [AHV95, Chapter 19].

Given a database **db**, the consistent answer to a first-order query $q(x_1, \ldots, x_n)$ is the set of tuples $(a_1, \ldots, a_n)$ such that every repair of **db** satisfies $q(a_1, \ldots, a_n)$.

From an incomplete database viewpoint, consistent answers are **certain answers**.

Database Repairs

A database **db** is consistent if $\mathbf{db} \models \Sigma$.

A repair of a (possibly inconsistent) database **db** is a consistent database that differs from **db** in a minimal way (which will be defined later).

Intuitively, we can view each repair as a **possible world**, which brings us in the realm of **incomplete databases** [IJ84], [AHV95, Chapter 19].

Given a database **db**, the consistent answer to a first-order query $q(x_1, \ldots, x_n)$ is the set of tuples $(a_1, \ldots, a_n)$ such that every repair of **db** satisfies $q(a_1, \ldots, a_n)$.

From an incomplete database viewpoint, consistent answers are **certain answers**.

Database Repairs

A database **db** is consistent if **db** $\models \Sigma$.

A repair of a (possibly inconsistent) database **db** is a consistent database that differs from **db** in a minimal way (which will be defined later).

Intuitively, we can view each repair as a **possible world**, which brings us in the realm of **incomplete databases** [IJ84], [AHV95, Chapter 19].

Given a database **db**, the consistent answer to a first-order query $q(x_1, \ldots, x_n)$ is the set of tuples $(a_1, \ldots, a_n)$ such that every repair of **db** satisfies $q(a_1, \ldots, a_n)$.

From an incomplete database viewpoint, consistent answers are **certain answers**.

Motivation from the Alice Book [AHV95]

19 Incomplete Information

Somebody: *What are we doing next?*

Alice: *Who are* we*? Who are* you*?*

Somebody: We *are you and the authors of the book, and* I *am one of them. This is an instance of incomplete information.*

Somebody: *It's not much, but we can still tell that* surely *one of us is Alice and that there are* possibly *up to three "Somebodies" speaking.*

In the previous parts, we have assumed that a database always records information that is completely known. Thus a database has consisted of a completely determined finite instance. In reality, we often must deal with incomplete information. This can be of many kinds. There can be missing information, as in "John bought a car but I don't know which one." In the case of John's car, the information exists but we do not have it. In other cases, some attributes may be relevant only to some tuples and irrelevant to others. Alice is single, so the spouse field is irrelevant in her case. Furthermore, some information may be imprecise: "Heather lives in a large and cheap apartment," where the values of *large* and *cheap* are fuzzy. Partial information may also arise when we cannot completely rely on the data because of possible inconsistencies (e.g., resulting from merging data from different sources).

Example (Database repairs)

Let $sort(R) = \{A, B\}$ and $\Sigma = \{R : A \to B\}$.

The following relation r has $2^n = \left(\sqrt{2}\right)^{|r|}$ repairs under all existing repair notions (see later).

r	A	B
	1	a
	1	b
	2	a
	2	b
	$\vdots$	
	n	a
	n	b

Table of Contents

Table of Contents

Functional Dependency (FD)

Syntax: $R : X \rightarrow Y$ with $X, Y \subseteq \mathit{sort}(R)$.

Semantics: **db** satisfies $R : X \rightarrow Y$ if for all $s, s' \in R^{\mathbf{db}}$,
if $s[X] = s'[X]$, then $s[Y] = s'[Y]$.

A key dependency is an FD $R : X \rightarrow Y$ with $Y = \mathit{sort}(R)$.

Inclusion Dependency (IND)

Syntax: $R[A_1, A_2, \ldots, A_m] \subseteq S[B_1, B_2, \ldots, B_m]$

- R, S are (possibly identical) relation names;
- $A_1, \ldots, A_m$ is a sequence of distinct attributes of $sort(R)$;
- $B_1, \ldots, B_m$ is a sequence of distinct attributes of $sort(S)$.

Semantics: satisfied by a database **db** if for every $s \in R^{\mathbf{db}}$, there exists some $s' \in R^{\mathbf{db}}$ such that for every $i \in \{1, \ldots, m\}$, $s(A_i) = s'(B_i)$.

Table of Contents

Tuple-Generating Dependency (tgd)

A (constant-free) database atom is an expression $R(x_1, \ldots, x_k)$ where R is a k-ary relation name and $x_1, \ldots, x_k$ are variables, not necessarily distinct.

Syntax of tgd: $\forall \vec{x} \, (\varphi(\vec{x}) \rightarrow \exists \vec{y} \, \psi(\vec{x}, \vec{y}))$

- φ and each ψ are conjunctions of database atoms;
- φ is not the empty conjunction (and thus the empty database satisfies every tgd);
- every variable in $\vec{x}$ appears in φ (but not necessarily in $\psi(\vec{x}, \vec{y})$).

Specializations:

- A tgd without existentially-quantified variables is called full.
- A LAV tgd is a tgd in which $\varphi(\vec{x})$ is a single database atom.

Tuple-Generating Dependency (tgd)

A (constant-free) database atom is an expression $R(x_1, \ldots, x_k)$ where R is a k-ary relation name and $x_1, \ldots, x_k$ are variables, not necessarily distinct.

Syntax of tgd: $\forall \vec{x}\, (\varphi(\vec{x}) \rightarrow \exists \vec{y}\, \psi(\vec{x}, \vec{y}))$

- φ and each ψ are conjunctions of database atoms;
- φ is not the empty conjunction (and thus the empty database satisfies every tgd);
- every variable in $\vec{x}$ appears in φ (but not necessarily in $\psi(\vec{x}, \vec{y})$).

Specializations:

- A tgd without existentially-quantified variables is called full.
- A LAV tgd is a tgd in which $\varphi(\vec{x})$ is a single database atom.

Examples

Example (Full tgd)

The binary relation R is transitive:

$$\forall x \forall y \forall z \left(\left(R(x, y) \land R(y, z) \right) \rightarrow R(x, z) \right)$$

Straightforward to chase, with termination.

Example (Non-full tgd)

Every path of length 2 extends to a path of length 4:

$$\forall x \forall y \forall z \left(\left(R(x, y) \land R(y, z) \right) \rightarrow \exists u \exists w \left(R(z, u) \land R(u, w) \right) \right)$$

Not a LAV, because the left-hand is not a lonely atom.

Example (LAV)

$$\forall y \forall z \left(R(y, z) \rightarrow \exists u \exists w \left(R(z, u) \land R(u, w) \right) \right)$$

Examples

Example (Full tgd)

The binary relation R is transitive:

$$\forall x \forall y \forall z \left(\left(R(x,y) \wedge R(y,z) \right) \rightarrow R(x,z) \right)$$

Straightforward to chase, with termination.

Example (Non-full tgd)

Every path of length 2 extends to a path of length 4:

$$\forall x \forall y \forall z \left(\left(R(x,y) \wedge R(y,z) \right) \rightarrow \exists u \exists w \left(R(z,u) \wedge R(u,w) \right) \right)$$

Not a LAV, because the left-hand is not a lonely atom.

Example (LAV)

$$\forall y \forall z \left(R(y,z) \rightarrow \exists u \exists w \left(R(z,u) \wedge R(u,w) \right) \right)$$

Examples

Example (Full tgd)

The binary relation R is transitive:

$$\forall x \forall y \forall z \left((R(x,y) \wedge R(y,z)) \rightarrow R(x,z) \right)$$

Straightforward to chase, with termination.

Example (Non-full tgd)

Every path of length 2 extends to a path of length 4:

$$\forall x \forall y \forall z \left((R(x,y) \wedge R(y,z)) \rightarrow \exists u \exists w \left(R(z,u) \wedge R(u,w) \right) \right)$$

Not a LAV, because the left-hand is not a lonely atom.

Example (LAV)

$$\forall y \forall z \left(R(y,z) \rightarrow \exists u \exists w \left(R(z,u) \wedge R(u,w) \right) \right)$$

Table of Contents

Universal Constraint (UC)

Syntax: $\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \wedge \beta(\vec{x}) \rightarrow S_1(\vec{y}_1) \vee \cdots \vee S_m(\vec{y}_m))$

- β is a Boolean combination of equalities;
- every variable in $\vec{x}$ appears in some $\vec{x}_i$ (but not necessarily in some $\vec{y}_j$).

Specializations:

- Denial constraint if $m = 0$:

$$\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \wedge \beta(\vec{x}) \rightarrow \textbf{false})$$

$$\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \rightarrow \neg\beta(x))$$

$$\neg\exists \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \wedge \beta(\vec{x}))$$

- Egd if $m = 0$ and β is the negation of an equality:

$$\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \wedge \neg\,(x_i = x_j) \rightarrow \textbf{false})$$

$$\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \rightarrow x_i = x_j)$$

Universal Constraint (UC)

Syntax: $\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \wedge \beta(\vec{x}) \rightarrow S_1(\vec{y}_1) \vee \cdots \vee S_m(\vec{y}_m))$

- β is a Boolean combination of equalities;
- every variable in $\vec{x}$ appears in some $\vec{x}_i$ (but not necessarily in some $\vec{y}_j$).

Specializations:

- Denial constraint if $m = 0$:

$$\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \wedge \beta(\vec{x}) \rightarrow \textbf{false})$$

$$\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \rightarrow \neg\beta(x))$$

$$\neg\exists \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \wedge \beta(\vec{x}))$$

- Egd if $m = 0$ and β is the negation of an equality:

$$\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \wedge \neg(x_i = x_j) \rightarrow \textbf{false})$$

$$\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \rightarrow x_i = x_j)$$

Universal Constraint (UC)

Syntax: $\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \wedge \beta(\vec{x}) \rightarrow S_1(\vec{y}_1) \vee \cdots \vee S_m(\vec{y}_m))$

- β is a Boolean combination of equalities;
- every variable in $\vec{x}$ appears in some $\vec{x}_i$ (but not necessarily in some $\vec{y}_j$).

Specializations:

- Denial constraint if $m = 0$:

$$\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \wedge \beta(\vec{x}) \rightarrow \textbf{false})$$

$$\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \rightarrow \neg\beta(x))$$

$$\neg\exists \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \wedge \beta(\vec{x}))$$

- Egd if $m = 0$ and β is the negation of an equality:

$$\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \wedge \neg\,(x_i = x_j) \rightarrow \textbf{false})$$

$$\forall \vec{x}\,(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \rightarrow x_i = x_j)$$

Example

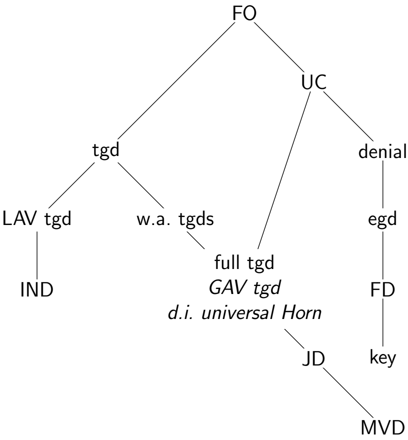
Example (Denial Constraint)

EMP	*Name*	*Rank*	*Sal*
	Ed	clerk	28
	Tim	clerk	30
	An	boss	40

No clerk earns more than any boss.

$$\neg \exists x_1 \exists x_2 \exists r_1 \exists r_2 \exists s_1 \exists s_2 \left(\begin{array}{l} \quad EMP(\underline{x_1}, r_1, s_1) \\ \wedge\, EMP(\underline{x_2}, r_2, s_2) \\ \wedge\, ((r_1 = \text{`clerk'}) \wedge (r_2 = \text{`boss'}) \wedge (s_1 > s_2)) \end{array} \right)$$

Overview



w.a. = weakly acyclic
d.i. = domain independent

The figure says, for example, that every set of FDs is logically equivalent to a set of egds.

Table of Contents

Minimal Difference (or Maximal Similarity)

We proceed in two steps:

1. For any arbitrary database **db**, define a binary relation $\preceq_{\mathbf{db}}$ on databases. Informally, $\mathbf{r} \preceq_{\mathbf{db}} \mathbf{s}$ means that

   r is more or equally similar to **db** than **s**.

2. Let Σ be a set of integrity constraints and **db** a database. We say that a database **r** is a repair of **db** with respect to Σ if
   - $\mathbf{r} \models \Sigma$, and
   - for every database **s**, if $\mathbf{s} \prec_{\mathbf{db}} \mathbf{r}$, then $\mathbf{s} \not\models \Sigma$.[1]

To guarantee the existence of repairs, it suffices to require that $\prec_{\mathbf{db}}$ be acyclic.

[1] $\mathbf{r} \prec_{\mathbf{db}} \mathbf{s}$ if ($\mathbf{r} \preceq_{\mathbf{db}} \mathbf{s}$ and not $\mathbf{s} \preceq_{\mathbf{db}} \mathbf{r}$).

Minimal Difference (or Maximal Similarity)

We proceed in two steps:

1. For any arbitrary database **db**, define a binary relation $\preceq_{\mathbf{db}}$ on databases. Informally, $\mathbf{r} \preceq_{\mathbf{db}} \mathbf{s}$ means that

   r is more or equally similar to **db** than **s**.

2. Let Σ be a set of integrity constraints and **db** a database.
   We say that a database **r** is a repair of **db** with respect to Σ if
   - $\mathbf{r} \models \Sigma$, and
   - for every database **s**, if $\mathbf{s} \prec_{\mathbf{db}} \mathbf{r}$, then $\mathbf{s} \not\models \Sigma$.[1]

To guarantee the existence of repairs, it suffices to require that $\prec_{\mathbf{db}}$ be acyclic.

[1] $\mathbf{r} \prec_{\mathbf{db}} \mathbf{s}$ if ($\mathbf{r} \preceq_{\mathbf{db}} \mathbf{s}$ and not $\mathbf{s} \preceq_{\mathbf{db}} \mathbf{r}$).

Table of Contents

Different Repair Notions

- For $\oplus$-repairs (Symmetric difference repairs), define:

$$\mathbf{r} \preceq_{\mathbf{db}} \mathbf{s} \text{ if } \quad \mathbf{r} \oplus \mathbf{db} \subseteq \mathbf{s} \oplus \mathbf{db}$$
$$\text{or, equivalently, } \mathbf{s} \cap \mathbf{db} \subseteq \mathbf{r} \subseteq \mathbf{s} \cup \mathbf{db}$$

  In this case, $\preceq_{\mathbf{db}}$ is a partial order (henceforth denoted $\preceq^{\oplus}_{\mathbf{db}}$).

- For C-repairs (Cardinality repairs), define:

$$\mathbf{r} \preceq_{\mathbf{db}} \mathbf{s} \text{ if } |\mathbf{r} \oplus \mathbf{db}| \leq |\mathbf{s} \oplus \mathbf{db}|.$$

  In this case, $\preceq_{\mathbf{db}}$ is a preorder[2] (henceforth denoted $\preceq^{C}_{\mathbf{db}}$).

- ...

Furthermore,

- a subset-repair is a $\oplus$-repair that is included in **db**; and
- a superset-repair is a $\oplus$-repair that includes **db**.

☞ For denial constraints, every $\oplus$-repair is a subset-repair.

[2] A preorder is reflexive and transitive.

Different Repair Notions

- For ⊕-repairs (Symmetric difference repairs), define:

$$\mathbf{r} \preceq_{\mathbf{db}} \mathbf{s} \text{ if } \quad \mathbf{r} \oplus \mathbf{db} \subseteq \mathbf{s} \oplus \mathbf{db}$$

$$\text{or, equivalently, } \mathbf{s} \cap \mathbf{db} \subseteq \mathbf{r} \subseteq \mathbf{s} \cup \mathbf{db}$$

  In this case, $\preceq_{\mathbf{db}}$ is a partial order (henceforth denoted $\preceq_{\mathbf{db}}^{\oplus}$).

- For C-repairs (Cardinality repairs), define:

$$\mathbf{r} \preceq_{\mathbf{db}} \mathbf{s} \text{ if } |\mathbf{r} \oplus \mathbf{db}| \leq |\mathbf{s} \oplus \mathbf{db}|.$$

  In this case, $\preceq_{\mathbf{db}}$ is a preorder[2] (henceforth denoted $\preceq_{\mathbf{db}}^{\mathsf{C}}$).

- ...

Furthermore,

- a subset-repair is a ⊕-repair that is included in **db**; and
- a superset-repair is a ⊕-repair that includes **db**.

☞ For denial constraints, every ⊕-repair is a subset-repair.

[2] A preorder is reflexive and transitive.

Examples of $\oplus$-Repair and C-Repair

Inconsistent database

EMP db

Name	*Rank*	*Sal*
Ed	clerk	28
Tim	clerk	30
An	boss	20
An	clerk	40

$$EMP : Name \rightarrow Rank, Sal$$

$$\forall^* \left(\left(\begin{array}{r} EMP(\underline{x_1}, \text{'clerk'}, s_1) \\ \wedge EMP(\underline{x_2}, \text{'boss'}, s_2) \end{array} \right) \rightarrow s_1 \leq s_2 \right)$$

Two symmetric difference repairs

EMP r

Name	*Rank*	*Sal*
Ed	clerk	28
Tim	clerk	30
An	clerk	40

and

EMP s

Name	*Rank*	*Sal*
An	boss	20

Only one cardinality repair

s is not a C-repair because $\mathbf{r} \prec^{C}_{\mathbf{db}} \mathbf{s}$.

Examples of $\oplus$-Repair and C-Repair

Inconsistent database

EMP **db**	$\underline{Name}$	$Rank$	Sal
	Ed	clerk	28
	Tim	clerk	30
	An	boss	20
	An	clerk	40

$$EMP : Name \rightarrow Rank, Sal$$

$$\forall^* \left(\left(\begin{array}{r} EMP(\underline{x_1}, \text{'clerk'}, s_1) \\ \wedge EMP(\underline{x_2}, \text{'boss'}, s_2) \end{array} \right) \rightarrow s_1 \leq s_2 \right)$$

Two symmetric difference repairs

EMP **r**	$\underline{Name}$	$Rank$	Sal
	Ed	clerk	28
	Tim	clerk	30
	An	clerk	40

and

EMP **s**	$\underline{Name}$	$Rank$	Sal
	An	boss	20

Only one cardinality repair

s is not a C-repair because $\mathbf{r} \prec^{C}_{\mathbf{db}} \mathbf{s}$.

Examples of $\oplus$-Repair and C-Repair

Inconsistent database

EMP **db**	<u>*Name*</u>	*Rank*	*Sal*
	Ed	clerk	28
	Tim	clerk	30
	An	boss	20
	An	clerk	40

$$EMP : Name \rightarrow Rank, Sal$$

$$\forall^* \left(\left(\begin{array}{r} EMP(\underline{x_1}, \text{'clerk'}, s_1) \\ \wedge EMP(\underline{x_2}, \text{'boss'}, s_2) \end{array} \right) \rightarrow s_1 \leq s_2 \right)$$

Two symmetric difference repairs

EMP **r**	<u>*Name*</u>	*Rank*	*Sal*
	Ed	clerk	28
	Tim	clerk	30
	An	clerk	40

and

EMP **s**	<u>*Name*</u>	*Rank*	*Sal*
	An	boss	20

Only one cardinality repair

s is not a C-repair because $\mathbf{r} \prec^{\mathsf{C}}_{\mathbf{db}} \mathbf{s}$.

Example (Inspired by [Ull88, p. 422])

$\color{red}{ENROLLED : Student, Prerequisite \to Year}$

$$\forall^* \left(\left(\begin{array}{l} ENROLLED(c,s,p,y) \\ \wedge\, ENROLLED(c,s',p',y') \end{array} \right) \to \exists z\, ENROLLED(c,s,p',z) \right)$$

ENROLLED	*Course*	*Student*	*Prerequisite*	*Year*	
	CS402	Jones	CS311	1988	(†)
	CS402	Jones	CS311	1989	(‡)
	CS402	Jones	CS401	1989	
	CS402	Smith	CS401	1989	

- The tuples † and ‡ together falsify the FD.
- If CS311 is a prerequisite of CS402, then Smith must have taken it.

Every $\oplus$-repair can be obtained in one of the following ways:

1. Delete both † and ‡.
2. Delete either † or ‡, and also delete the tuple about Smith.
3. Delete either † or ‡, and insert (CS402, Smith, CS311, 19xx) for some year 19xx.

Example (Inspired by [Ull88, p. 422])

$\mathit{ENROLLED} : \mathit{Student}, \mathit{Prerequisite} \rightarrow \mathit{Year}$

$$\forall^* \left(\left(\begin{array}{l} \mathit{ENROLLED}(c, s, p, y) \\ \wedge\, \mathit{ENROLLED}(c, s', p', y') \end{array} \right) \rightarrow \exists z\, \mathit{ENROLLED}(c, s, p', z) \right)$$

ENROLLED	*Course*	*Student*	*Prerequisite*	*Year*	
	CS402	Jones	CS311	1988	(†)
	CS402	Jones	CS311	1989	(‡)
	CS402	Jones	CS401	1989	
	CS402	Smith	CS401	1989	

- The tuples † and ‡ together falsify the FD.
- If CS311 is a prerequisite of CS402, then Smith must have taken it.

Every ⊕-repair can be obtained in one of the following ways:

1. Delete both † and ‡.
2. Delete either † or ‡, and also delete the tuple about Smith.
3. Delete either † or ‡, and insert (CS402, Smith, CS311, 19xx) for some year 19xx.

Example (Inspired by [Ull88, p. 422])

$ENROLLED : Student, Prerequisite \rightarrow Year$

$$\forall^* \left(\left(\begin{array}{l} ENROLLED(c,s,p,y) \\ \wedge\, ENROLLED(c,s',p',y') \end{array} \right) \rightarrow \exists z\, ENROLLED(c,s,p',z) \right)$$

ENROLLED	*Course*	*Student*	*Prerequisite*	*Year*	
	CS402	Jones	CS311	1988	(†)
	CS402	Jones	CS311	1989	(‡)
	CS402	Jones	CS401	1989	
	CS402	Smith	CS401	1989	

- The tuples † and ‡ together falsify the FD.
- If CS311 is a prerequisite of CS402, then Smith must have taken it.

Every ⊕-repair can be obtained in one of the following ways:

1. Delete both † and ‡.
2. Delete either † or ‡, and also delete the tuple about Smith.
3. Delete either † or ‡, and insert (CS402, Smith, CS311, 19xx) for some year 19xx.

Example (Inspired by [Ull88, p. 422])

ENROLLED : *Student*, *Prerequisite* → *Year*

$$\forall^* \left(\left(\begin{array}{l} ENROLLED(c, s, p, y) \\ \wedge\, ENROLLED(c, s', p', y') \end{array} \right) \rightarrow \exists z\, ENROLLED(c, s, p', z) \right)$$

ENROLLED	*Course*	*Student*	*Prerequisite*	*Year*	
	CS402	Jones	CS311	1988	(†)
	CS402	Jones	CS311	1989	(‡)
	CS402	Jones	CS401	1989	
	CS402	Smith	CS401	1989	

- The tuples † and ‡ together falsify the FD.
- If CS311 is a prerequisite of CS402, then Smith must have taken it.

Every ⊕-repair can be obtained in one of the following ways:

1. Delete both † and ‡.
2. Delete either † or ‡, and also delete the tuple about Smith.
3. Delete either † or ‡, and insert (CS402, Smith, CS311, 19xx) for some year 19xx.

Adapted from Example 3.3 in [HW22]

$$\forall x \forall y \left(R(\underline{x}, y) \rightarrow \exists z\, S(\underline{y}, z)\right) \quad \text{and} \quad \forall y \forall z \left(S(\underline{y}, z) \rightarrow T(\underline{z})\right).$$

$$\mathbf{db} = \quad \begin{array}{c|cc} R & \underline{x} & y \\ \hline & a & b \end{array} \quad \begin{array}{c|cc} S & \underline{y} & z \\ \hline & b & c \end{array} \quad \begin{array}{c|c} T & \underline{z} \\ \hline & \end{array}$$

$$\mathbf{r} = \quad \begin{array}{c|cc} R & \underline{x} & y \\ \hline & a & b \end{array} \quad \begin{array}{c|cc} S & \underline{y} & z \\ \hline & b & c \end{array} \quad \begin{array}{c|c} T & \underline{z} \\ \hline & c \end{array}$$

$$\mathbf{s} = \quad \begin{array}{c|cc} R & \underline{x} & y \\ \hline & a & b \end{array} \quad \begin{array}{c|cc} S & \underline{y} & z \\ \hline & b & \perp \end{array} \quad \begin{array}{c|c} T & \underline{z} \\ \hline & \perp \end{array}$$

$$\begin{aligned} \mathbf{r} \oplus \mathbf{db} &= \{T(\underline{c})\} \\ \mathbf{s} \oplus \mathbf{db} &= \{S(\underline{b}, c), S(\underline{b}, \perp), T(\underline{\perp})\} \end{aligned}$$

Note that $\mathbf{r} \oplus \mathbf{db}$ and $\mathbf{s} \oplus \mathbf{db}$ are not comparable by set inclusion.
This implies, in particular, $\mathbf{r} \not\preceq^{\oplus}_{\mathbf{db}} \mathbf{s}$.

Adapted from Example 3.3 in [HW22]

$$\forall x \forall y \left(R(\underline{x}, y) \rightarrow \exists z\, S(\underline{y}, z)\right) \quad \text{and} \quad \forall y \forall z \left(S(\underline{y}, z) \rightarrow T(\underline{z})\right).$$

db =

R	$\underline{x}$	y
	a	b

S	$\underline{y}$	z
	b	c

T	$\underline{z}$

r =

R	$\underline{x}$	y
	a	b

S	$\underline{y}$	z
	b	c

T	$\underline{z}$
	c

s =

R	$\underline{x}$	y
	a	b

S	$\underline{y}$	z
	b	$\perp$

T	$\underline{z}$
	$\perp$

$$\begin{aligned} \mathbf{r} \oplus \mathbf{db} &= \{T(\underline{c})\} \\ \mathbf{s} \oplus \mathbf{db} &= \{S(\underline{b}, c), S(\underline{b}, \perp), T(\underline{\perp})\} \end{aligned}$$

Note that $\mathbf{r} \oplus \mathbf{db}$ and $\mathbf{s} \oplus \mathbf{db}$ are not comparable by set inclusion. This implies, in particular, $\mathbf{r} \not\preceq^{\oplus}_{\mathbf{db}} \mathbf{s}$.

Adapted from Example 3.3 in [HW22]

$\forall x \forall y \left(R(\underline{x}, y) \rightarrow \exists z\, S(\underline{y}, z)\right)$ and $\forall y \forall z \left(S(\underline{y}, z) \rightarrow T(\underline{z})\right)$.

$$\mathbf{db} = \quad \begin{array}{c|cc} R & \underline{x} & y \\ \hline & a & b \end{array} \quad \begin{array}{c|cc} S & \underline{y} & z \\ \hline & b & c \end{array} \quad \begin{array}{c|c} T & \underline{z} \\ \hline & \end{array}$$

$$\mathbf{r} = \quad \begin{array}{c|cc} R & \underline{x} & y \\ \hline & a & b \end{array} \quad \begin{array}{c|cc} S & \underline{y} & z \\ \hline & b & c \end{array} \quad \begin{array}{c|c} T & \underline{z} \\ \hline & c \end{array}$$

$$\mathbf{s} = \quad \begin{array}{c|cc} R & \underline{x} & y \\ \hline & a & b \end{array} \quad \begin{array}{c|cc} S & \underline{y} & z \\ \hline & b & \perp \end{array} \quad \begin{array}{c|c} T & \underline{z} \\ \hline & \perp \end{array}$$

$$\begin{aligned} \mathbf{r} \oplus \mathbf{db} &= \{T(\underline{c})\} \\ \mathbf{s} \oplus \mathbf{db} &= \{S(\underline{b}, c), S(\underline{b}, \perp), T(\underline{\perp})\} \end{aligned}$$

Note that $\mathbf{r} \oplus \mathbf{db}$ and $\mathbf{s} \oplus \mathbf{db}$ are not comparable by set inclusion.
This implies, in particular, $\mathbf{r} \not\preceq^{\oplus}_{\mathbf{db}} \mathbf{s}$.

Loosely Sound Semantics [CLR03]

Maximize the set of preserved database facts (i.e., minimize deletions), while considering that insertions are harmless.

- For loosely sound semantics, define:

$$\mathbf{r} \preceq_{\mathbf{db}} \mathbf{s} \text{ if } \mathbf{r} \cap \mathbf{db} \supseteq \mathbf{s} \cap \mathbf{db}.$$

In this case, $\preceq_{\mathbf{db}}$ is a preorder.

Open World Assumption: Add as much as you like.

☞ Having repairs $\mathbf{r}$ and $\mathbf{r} \uplus \Delta$, with Δ a "superfluous" addition, is harmless for CQA to conjunctive queries q, because

$$q(\mathbf{r}) \cap q(\mathbf{r} \cup \Delta) = q(\mathbf{r}).$$

Loosely Sound Semantics [CLR03]

Maximize the set of preserved database facts (i.e., minimize deletions), while considering that insertions are harmless.

- For loosely sound semantics, define:

$$\mathbf{r} \preceq_{\mathbf{db}} \mathbf{s} \text{ if } \mathbf{r} \cap \mathbf{db} \supseteq \mathbf{s} \cap \mathbf{db}.$$

  In this case, $\preceq_{\mathbf{db}}$ is a preorder.

Open World Assumption: Add as much as you like.

☞ Having repairs $\mathbf{r}$ and $\mathbf{r} \uplus \Delta$, with Δ a "superfluous" addition, is harmless for CQA to conjunctive queries q, because

$$q(\mathbf{r}) \cap q(\mathbf{r} \cup \Delta) = q(\mathbf{r}).$$

Another Natural but Unexplored (?) Repair Notion

Loosely Sound Semantics + "minimal insertions"

- For max-intersection-repairs, define:

  $\mathbf{r} \preceq_{\mathbf{db}} \mathbf{s}$ *if either*

  - $\mathbf{r} \cap \mathbf{db} \supsetneq \mathbf{s} \cap \mathbf{db}$*; or*
  - *both* $\mathbf{r} \cap \mathbf{db} = \mathbf{s} \cap \mathbf{db}$ *and* $\mathbf{r} \subseteq \mathbf{s}$.

  In this case, $\preceq_{\mathbf{db}}$ is a partial order.

Proposition

Every max-intersection-repair is a $\oplus$-repair.

☞ For tgds, every max-intersection-repair is a superset-repair.

Another Natural but Unexplored (?) Repair Notion

Loosely Sound Semantics + "minimal insertions"

- For max-intersection-repairs, define:

  $\mathbf{r} \preceq_{\mathbf{db}} \mathbf{s}$ *if either*

  - $\mathbf{r} \cap \mathbf{db} \supsetneq \mathbf{s} \cap \mathbf{db}$; *or*
  - *both* $\mathbf{r} \cap \mathbf{db} = \mathbf{s} \cap \mathbf{db}$ *and* $\mathbf{r} \subseteq \mathbf{s}$.

  In this case, $\preceq_{\mathbf{db}}$ is a partial order.

Proposition

Every max-intersection-repair is a $\oplus$-repair.

☞ For tgds, every max-intersection-repair is a superset-repair.

Another Natural but Unexplored (?) Repair Notion

Loosely Sound Semantics + "minimal insertions"

- For max-intersection-repairs, define:

  $\mathbf{r} \preceq_{\mathbf{db}} \mathbf{s}$ *if either*

  - $\mathbf{r} \cap \mathbf{db} \supsetneq \mathbf{s} \cap \mathbf{db}$; *or*
  - *both* $\mathbf{r} \cap \mathbf{db} = \mathbf{s} \cap \mathbf{db}$ *and* $\mathbf{r} \subseteq \mathbf{s}$.

  In this case, $\preceq_{\mathbf{db}}$ is a partial order.

Proposition

Every max-intersection-repair is a $\oplus$-repair.

☞ For tgds, every max-intersection-repair is a superset-repair.

Example (Max-intersection-repair)

ENROLLED	*Course*	*Student*	*Prerequisite*	*Year*	
	CS402	Jones	CS311	1988	(†)
	CS402	Jones	CS311	1989	(‡)
	CS402	Jones	CS401	1989	
	CS402	Smith	CS401	1989	

Every max-intersection-repair can be obtained as follows:

- Delete either † or ‡, and insert (CS402, Smith, CS311, 19xx) for some year 19xx.

Table of Contents

Example (Information loss in tuple-based repairing)

$$\Sigma = \left\{ \begin{array}{l} \mathit{TAUGHT\text{-}BY} : \mathit{Course} \rightarrow \mathit{Teacher}, \\ \mathit{TAUGHT\text{-}BY} : \mathit{Teacher}, \mathit{Hour} \rightarrow \mathit{Course} \end{array} \right\}$$

TAUGHT-BY **db**

Course	*Teacher*	*Hour*
CS402	D. Maier	Mon. 10am
CS402	J. Ullman	Fri. 10am

TAUGHT-BY **r**

Course	*Teacher*	*Hour*
CS402	D. Maier	Mon. 10am

TAUGHT-BY How about **s**?

Course	*Teacher*	*Hour*
CS402	D. Maier	Mon. 10am
CS402	D. Maier	Fri. 10am

- $\mathbf{s} \not\preceq_{\mathbf{db}} \mathbf{r}$ for all the previously proposed tuple-based preorders $\preceq_{\mathbf{db}}$.
- Yet one may feel that **s** is "better" than **r**, because it also preserves $\exists x\ \mathit{TAUGHT\text{-}BY}(\text{CS402}, x, \text{Fri. 10am})$.
- See [Wij05] for a logical approach to value-based repairing.

Example (Information loss in tuple-based repairing)

$$\Sigma = \left\{ \begin{array}{l} \mathit{TAUGHT\text{-}BY} : \mathit{Course} \rightarrow \mathit{Teacher}, \\ \mathit{TAUGHT\text{-}BY} : \mathit{Teacher}, \mathit{Hour} \rightarrow \mathit{Course} \end{array} \right\}$$

TAUGHT-BY **db**

Course	*Teacher*	*Hour*
CS402	D. Maier	Mon. 10am
CS402	J. Ullman	Fri. 10am

TAUGHT-BY **r**

Course	*Teacher*	*Hour*
CS402	D. Maier	Mon. 10am

TAUGHT-BY How about **s**?

Course	*Teacher*	*Hour*
CS402	D. Maier	Mon. 10am
CS402	D. Maier	Fri. 10am

- $\mathbf{s} \not\preceq_{\mathbf{db}} \mathbf{r}$ for all the previously proposed tuple-based preorders $\preceq_{\mathbf{db}}$.
- Yet one may feel that **s** is "better" than **r**, because it also preserves $\exists x\, \mathit{TAUGHT\text{-}BY}(\mathrm{CS402}, x, \mathrm{Fri.\ 10am})$.
- See [Wij05] for a logical approach to value-based repairing.

Example (Information loss in tuple-based repairing)

$$\Sigma = \left\{ \begin{array}{l} \mathit{TAUGHT\text{-}BY} : \mathit{Course} \rightarrow \mathit{Teacher}, \\ \mathit{TAUGHT\text{-}BY} : \mathit{Teacher}, \mathit{Hour} \rightarrow \mathit{Course} \end{array} \right\}$$

TAUGHT-BY **db**	*Course*	*Teacher*	*Hour*
	CS402	D. Maier	Mon. 10am
	CS402	J. Ullman	Fri. 10am

TAUGHT-BY **r**	*Course*	*Teacher*	*Hour*
	CS402	D. Maier	Mon. 10am

TAUGHT-BY How about **s**?	*Course*	*Teacher*	*Hour*
	CS402	D. Maier	Mon. 10am
	CS402	D. Maier	Fri. 10am

- **s** $\not\preceq_{\mathbf{db}}$ **r** for all the previously proposed tuple-based preorders $\preceq_{\mathbf{db}}$.
- Yet one may feel that **s** is "better" than **r**, because it also preserves $\exists x\ \mathit{TAUGHT\text{-}BY}(\text{CS402}, x, \text{Fri. 10am})$.
- See [Wij05] for a logical approach to value-based repairing.

Homomorphism-based Repairs

Example

db	*Name*	*Rank*	*Sal*
	Ed	clerk	28
	Tim	clerk	30
	An	boss	20
	An	clerk	40

rep	*Name*	*Rank*	*Sal*
	Ed	clerk	28
	Tim	clerk	30
	An	boss	40

glb(**rep**, **db**)	*Name*	*Rank*	*Sal*
	Ed	clerk	28
	Tim	clerk	30
	An	boss	y
	An	x	40

rep $\cap$ **db**	*Name*	*Rank*	*Sal*
	Ed	clerk	28
	Tim	clerk	30

Repairing Numerical Attributes

Assumptions

- Primary keys are satisfied and immutable.
- Inconsistencies in numerical data.

Inconsistent numerical data

$\forall x \forall y \forall z \, (EMP(\underline{x}, y, z) \wedge (y < 5) \rightarrow (z \leq 6000))$

EMP	Emp	Status	Sal
	Ed	2	6100
	Tim	4	9000

Two approaches

- Update Based [FFP10]
- Least Square Fixes [BBFL08]

Update Based [FFP10]

Principle "update based"

- **r** is preferred to **s** if it requires updating a smaller set of values (in terms of set inclusion or cardinality).
- The actual new values after update do not matter.

Update based

$\forall x \forall y \forall z \, (EMP(\underline{x}, y, z) \wedge (y < 5) \rightarrow (z \leq 6000))$

EMP	<u>Emp</u>	Status	Sal
t_1	Ed	2	6100
t_2	Tim	4	9000

$\rightsquigarrow$

<u>Emp</u>	Status	Sal
Ed	8	6100
Tim	4	1000

The atomic updates are $(t_1, Status, \text{'8'})$ and $(t_2, Sal, \text{'1000'})$.
The set of updated values is $\{(t_1, Status), (t_2, Sal)\}$.

Least Square Fixes [BBFL08]

Principle "least square fixes"

r is preferred to **s** if the distance between **r** and **db** is smaller than the distance between **s** and **db**.

Least square fixes

$\forall x \forall y \forall z \, (EMP(\underline{x}, y, z) \wedge (y < 5) \rightarrow (z \leq 6000))$

EMP

Emp	Status	Sal
Ed	2	6100
Tim	4	9000

$\rightsquigarrow$

Emp	Status	Sal
Ed	2	6000
Tim	5	9000

Distance for Ed-tuple:	$w_{\mathrm{Status}}(2-2)^2 + w_{\mathrm{Sal}}(6100-6000)^2$
Distance for Tim-tuple:	$w_{\mathrm{Status}}(4-5)^2 + w_{\mathrm{Sal}}(9000-9000)^2$
Global distance:	Σ

Comparison Repairing Numercial Attributes

Repairs are different in both approaches.

Comparison

$\forall x \forall y \forall z\, (ECTS(\underline{x}, y, z) \rightarrow (y + z \geq 120))$

ECTS

<u>SID</u>	Year1	Year2
Ed	68	60
Tim	59	59

$\rightsquigarrow$

<u>SID</u>	Year1	Year2
Ed	68	60
Tim	60	60

This would not be a prefered repair in the update based approach.

Table of Contents

Data Complexity of Repair Checking

Definition (Repair checking)

For a finite set Σ of integrity constraints, $\oplus$-RC(Σ) is the following problem:

INSTANCE: Databases **db** and **r** (over a fixed schema).

QUESTION: Is **r** a $\oplus$-repair of **db**?

Definition

Let *IC* be a class of integrity constraints (for example, FDs, INDs...). Let **C** be a complexity class. The $\oplus$-repair checking problem for *IC* is said to be

Upper bound: in **C** if for every finite subset Σ of *IC*, $\oplus$-RC(Σ) is in **C**.

Upper+Lower bound: **C**-complete if it is in **C** and **there exists** a finite subset Σ of *IC* such that $\oplus$-RC(Σ) is **C**-complete.

Data Complexity of Repair Checking

Definition (Repair checking)

For a finite set Σ of integrity constraints, $\oplus$-RC(Σ) is the following problem:

INSTANCE: Databases **db** and **r** (over a fixed schema).

QUESTION: Is **r** a $\oplus$-repair of **db**?

Definition

Let *IC* be a class of integrity constraints (for example, FDs, INDs...). Let **C** be a complexity class. The $\oplus$-repair checking problem for *IC* is said to be

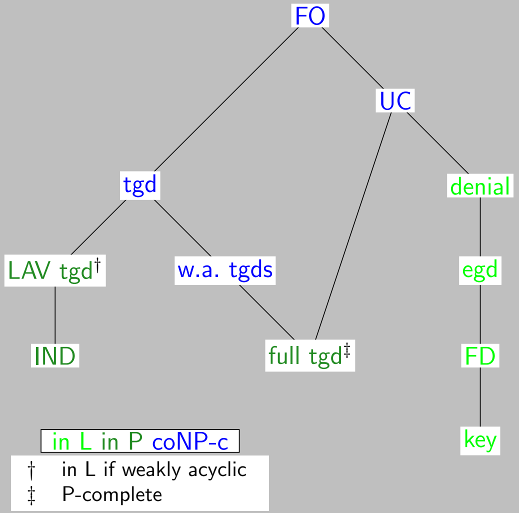
Upper bound: in **C** if for every finite subset Σ of *IC*, $\oplus$-RC(Σ) is in **C**.

Upper+Lower bound: **C**-complete if it is in **C** and **there exists** a finite subset Σ of *IC* such that $\oplus$-RC(Σ) is **C**-complete.

Recall of Complexity Classes

- **FO**, the class of decision problems (in this case, sets of structures) that can be defined in first-order logic (descriptive complexity).
- **L**, the class of decision problems that can be solved in deterministic logarithmic space.
- **P**, the class of decision problems that can be solved in deterministic polynomial time.
- **NP**, the class of decision problems whose "yes" instances have succinct certificates that can be verified in deterministic polynomial time.
- **coNP**, the class of decision problems whose "no" instances have succinct disqualifications that can be verified in deterministic polynomial time.
- **FP**, the class of function problems that can be solved in deterministic polynomial time.
- ♯**P**, the class of counting problems associated with decision problems in **NP**. Given an instance of a decision problem in **NP**, the associated counting problem instance asks to determine the number of succinct certificates of its being a "yes" instance.

Complexity of ⊕-Repair Checking



Proposition ([AK09])

If Σ is a finite set of first-order constraints, then $\oplus$-RC(Σ) is in **coNP**.

Proof sketch.

Let Σ be a finite set of first-order constraints. Assume that $(\mathbf{db}, \mathbf{r})$ is a "no"-instance of $\oplus$-RC(Σ).

- If $\mathbf{r} \not\models \Sigma$, which can be tested in polynomial time, then $\mathbf{r}$ cannot be a repair.
- Assume $\mathbf{r} \models \Sigma$ from here on. Then, there is a repair $\mathbf{s}$ of $\mathbf{db}$ such that $\mathbf{s} \prec^{\oplus}_{\mathbf{db}} \mathbf{r}$. Consequently, $\mathbf{s} \subseteq \mathbf{r} \cup \mathbf{db}$, and hence $\mathbf{s}$ is of polynomial size. It can be checked in polynomial time that $\mathbf{s} \oplus \mathbf{db} \subsetneq \mathbf{r} \oplus \mathbf{db}$ and $\mathbf{s} \models \Sigma$.

It is now correct to conclude that "no"-instances have "succinct disqualifications." $\square$

Proposition ([AK09])

If Σ is a finite set of first-order constraints, then $\oplus$-RC(Σ) is in **coNP***.*

Proof sketch.

Let Σ be a finite set of first-order constraints. Assume that $(\mathbf{db}, \mathbf{r})$ is a "no"-instance of $\oplus$-RC(Σ).

- If $\mathbf{r} \not\models \Sigma$, which can be tested in polynomial time, then $\mathbf{r}$ cannot be a repair.
- Assume $\mathbf{r} \models \Sigma$ from here on. Then, there is a repair $\mathbf{s}$ of $\mathbf{db}$ such that $\mathbf{s} \prec^{\oplus}_{\mathbf{db}} \mathbf{r}$. Consequently, $\mathbf{s} \subseteq \mathbf{r} \cup \mathbf{db}$, and hence $\mathbf{s}$ is of polynomial size. It can be checked in polynomial time that $\mathbf{s} \oplus \mathbf{db} \subsetneq \mathbf{r} \oplus \mathbf{db}$ and $\mathbf{s} \models \Sigma$.

It is now correct to conclude that "no"-instances have "succinct disqualifications." □

Proposition ([AK09])

If Σ is a finite set of denial constraints, then $\oplus$-RC(Σ) is in **L**.

Proof sketch.

Let Σ be a finite set of denial constraints. Let $(\mathbf{db}, \mathbf{r})$ be an instance of $\oplus$-RC(Σ). Note that every $\oplus$-repair w.r.t. denial constraints is a subset-repair. The following are equivalent:

1. $\mathbf{r}$ is a $\oplus$-repair; and
2. $\mathbf{r} \models \Sigma$, $\mathbf{r} \subseteq \mathbf{db}$, and for every $A \in \mathbf{db} \setminus \mathbf{r}$, we have that $\mathbf{r} \cup \{A\}$ is inconsistent. These conditions can be checked in logarithmic space.

$\square$

Proposition ([AK09])

If Σ is a finite set of denial constraints, then $\oplus$-RC(Σ) is in $\mathbf{L}$.

Proof sketch.

Let Σ be a finite set of denial constraints. Let $(\mathbf{db}, \mathbf{r})$ be an instance of $\oplus$-RC(Σ). Note that every $\oplus$-repair w.r.t. denial constraints is a subset-repair. The following are equivalent:

1. $\mathbf{r}$ is a $\oplus$-repair; and
2. $\mathbf{r} \models \Sigma$, $\mathbf{r} \subseteq \mathbf{db}$, and for every $A \in \mathbf{db} \setminus \mathbf{r}$, we have that $\mathbf{r} \cup \{A\}$ is inconsistent. These conditions can be checked in logarithmic space.

□

Open Challenges

Combining Classes of Integrity Constraints

Most real-life databases have constraints from different classes. Therefore, it seems normal to consider unions of classes of integrity constraints.

Proposition

The $\oplus$-repair checking problem for FDs and INDs taken together is **coNP***-complete.*

Fine-Grained Complexity Classification

Assume that $\oplus$-repair checking for a class IC is **coNP**-complete. It is normal to ask whether the set

$$\{\oplus\text{-RC}(\Sigma) \mid \Sigma \text{ is a finite subset of } IC\}$$

exhibits an effective complexity dichotomy between **P** and **coNP**-complete. Such questions remain largely unanswered.

Open Challenges

Combining Classes of Integrity Constraints

Most real-life databases have constraints from different classes. Therefore, it seems normal to consider unions of classes of integrity constraints.

Proposition

The $\oplus$-repair checking problem for FDs and INDs taken together is **coNP***-complete.*

Fine-Grained Complexity Classification

Assume that $\oplus$-repair checking for a class IC is **coNP**-complete. It is normal to ask whether the set

$$\{\oplus\text{-RC}(\Sigma) \mid \Sigma \text{ is a finite subset of } IC\}$$

exhibits an effective complexity dichotomy between **P** and **coNP**-complete. Such questions remain largely unanswered.

Table of Contents

Conjunctive Queries

Conjunctive queries are first-order formulas of the form

$$\exists \vec{x} \left(R_1(\vec{x}_1) \wedge \cdots \wedge R_n(\vec{x}_n) \right).$$

Such conjunctive query is self-join-free if $R_i \neq R_j$ whenever $i \neq j$.

A conjunctive query is Boolean if it contains no free variables.

Data Complexity of Consistent Conj. Query Answering

Definition (Consistent query answering)

For a finite set Σ of integrity constraints and a Boolean query q, we define $\oplus$-CQA(Σ, q) as the following problem:

INSTANCE: Database **db** (over a fixed schema).

QUESTION: Is q true in every $\oplus$-repair of **db**?

Definition

Let IC be a class of integrity constraints (for example, FDs, INDs...). Let **C** be a complexity class. The $\oplus$-consistent **conjunctive** query answering problem for IC is said to be

Upper bound: in **C** if for every finite subset Σ of IC and Boolean conjunctive query q, $\oplus$-CQA(Σ, q) is in **C**.

Upper+Lower bound: **C**-complete if it is in **C** and **there exist** a finite subset Σ of IC and a Boolean conjunctive query q such that $\oplus$-CQA(Σ, q) is **C**-complete.

Data Complexity of Consistent Conj. Query Answering

Definition (Consistent query answering)

For a finite set Σ of integrity constraints and a Boolean query q, we define $\oplus$-CQA(Σ, q) as the following problem:

INSTANCE: Database **db** (over a fixed schema).

QUESTION: Is q true in every $\oplus$-repair of **db**?

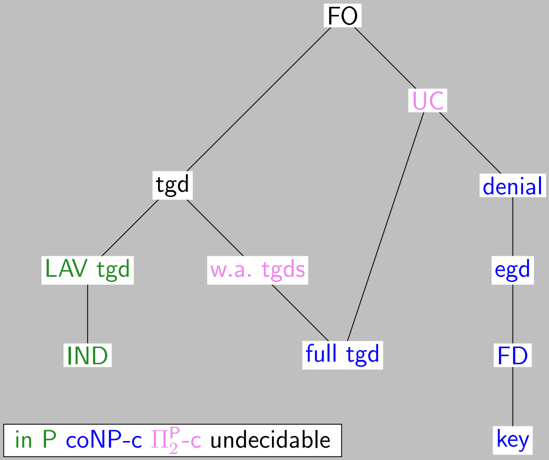
Definition

Let *IC* be a class of integrity constraints (for example, FDs, INDs...). Let **C** be a complexity class. The $\oplus$-consistent **conjunctive** query answering problem for *IC* is said to be

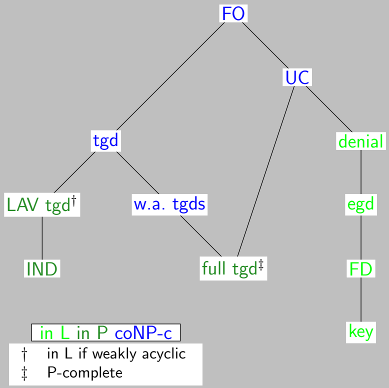
Upper bound: in **C** if for every finite subset Σ of *IC* and Boolean conjunctive query q, $\oplus$-CQA(Σ, q) is in **C**.

Upper+Lower bound: **C**-complete if it is in **C** and **there exist** a finite subset Σ of *IC* and a Boolean conjunctive query q such that $\oplus$-CQA(Σ, q) is **C**-complete.

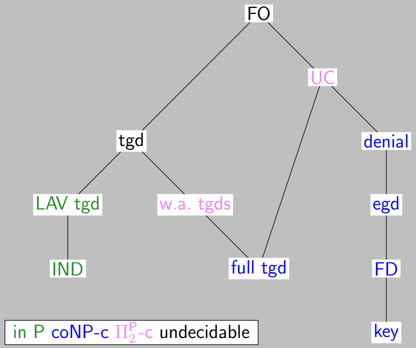
Complexity of $\oplus$-Consistent Conj. Query Answering



⊕-repair checking



⊕-CCQA



Discussion

Let Σ be a finite set of full tgds.

There is a polynomial time algorithm ("the chase") that, given a database **db**, computes the unique superset-repair of **db** with respect to Σ.

☞ $\oplus$-consistent conjunctive query answering for full tgds is intractable because tuple deletions are on equal footing with tuple insertions.

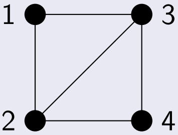
coNP-Hardness for Key

Proposition

$\oplus$-CQA(Σ, q) *is* **coNP**-*hard for* $\Sigma = \{C : \mathit{Vertex} \rightarrow \mathit{Color}\}$ *and* $q = \exists x \exists y \exists z \left(C(\underline{x}, z) \wedge C(\underline{y}, z) \wedge E(\underline{x, y}) \right)$.

Proof sketch.

Graph is 3-colorable $\iff$ q is false in some repair



C	<u>*Vertex*</u>	*Color*
	1	'red'
	1	'blue'
	1	'yellow'
	$\vdots$	$\vdots$
	4	'red'
	4	'blue'
	4	'yellow'

E	<u>*From*</u>	<u>*To*</u>
	1	2
	1	3
	2	3
	2	4
	3	4

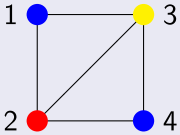
coNP-Hardness for Key

Proposition

$\oplus$-CQA(Σ, q) *is* **coNP**-*hard for* $\Sigma = \{C : \mathit{Vertex} \rightarrow \mathit{Color}\}$ *and*
$q = \exists x \exists y \exists z \left(C(\underline{x}, z) \wedge C(\underline{y}, z) \wedge E(\underline{x, y}) \right)$.

Proof sketch.

Graph is 3-colorable $\iff$ q is false in some repair



C	$\underline{\mathit{Vertex}}$	Color
	1	'blue'
	2	'red'
	3	'yellow'
	4	'blue'

E	$\underline{\mathit{From}}$	$\underline{\mathit{To}}$
	1	2
	1	3
	2	3
	2	4
	3	4

□

Open Challenges

Combining Classes of Integrity Constraints

For example, primary keys and foreign keys (or, more generally, INDs).

Fine-Grained Complexity Classification

Assume that $\oplus$-consistent conj. query answering for a class IC is **coNP**-complete. It is normal to ask whether the set

$$\{ \oplus\text{-CQA}(\Sigma, q) \mid \begin{array}{l} \Sigma \text{ is a finite subset of } IC \text{ and} \\ q \text{ is a Boolean conjunctive query} \end{array} \}$$

exhibits an effective complexity dichotomy between **P** and **coNP**-complete.

Recall the point of dichotomy theorems:

If $\mathbf{P} \neq \mathbf{NP}$,
then **NPI** is nonempty.

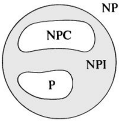


Figure 7.1 The world of NP, reprised (assuming $P \neq NP$).

Table of Contents

Preliminaries

We will assume that every relation name R is associated with some arity n and a (primary) key dependency $\{1, \ldots, k\} \rightarrow \{1, \ldots, n\}$ ($k \leq n$). In this case we say that R has signature $[n, k]$. We thus assume that all primary-key positions precede all non-primary-key positions.

From here on, by key we mean the primary key $\{1, \ldots, k\}$.

Let R be a relation name with signature $[n, k]$. An R-atom takes the form $R(\underline{s_1, \ldots, s_k}, s_{k+1}, \ldots, s_n)$, where each s_i is a variable or a constant. An R-fact is an R-atom without variables. Two facts are key-equal if they agree on their relation names and on all key positions. For example, $S(\underline{a, b}, d)$ and $S(\underline{a, b}, e)$ are key-equal.

A database (instance) **db** is a finite set of facts. A block in **db** is a maximal set of key-equal facts. Blocks will be separated by dashed lines. A relation in **db** is a maximal set of facts with the same relation name. A database **db** is consistent if it contains no block with two or more facts. A repair of **db** selects exactly one fact from each block.

We will assume that every relation name R is associated with some arity n and a (primary) key dependency $\{1, \ldots, k\} \rightarrow \{1, \ldots, n\}$ $(k \leq n)$. In this case we say that R has signature $[n, k]$. We thus assume that all primary-key positions precede all non-primary-key positions.

From here on, by key we mean the primary key $\{1, \ldots, k\}$.

Let R be a relation name with signature $[n, k]$. An R-atom takes the form $R(\underline{s_1, \ldots, s_k}, s_{k+1}, \ldots, s_n)$, where each s_i is a variable or a constant. An R-fact is an R-atom without variables. Two facts are key-equal if they agree on their relation names and on all key positions. For example, $S(\underline{a, b}, d)$ and $S(\underline{a, b}, e)$ are key-equal.

A database (instance) **db** is a finite set of facts. A block in **db** is a maximal set of key-equal facts. Blocks will be separated by dashed lines. A relation in **db** is a maximal set of facts with the same relation name. A database **db** is consistent if it contains no block with two or more facts. A repair of **db** selects exactly one fact from each block.

Preliminaries

We will assume that every relation name R is associated with some arity n and a (primary) key dependency $\{1, \ldots, k\} \rightarrow \{1, \ldots, n\}$ $(k \leq n)$. In this case we say that R has signature $[n, k]$. We thus assume that all primary-key positions precede all non-primary-key positions.

From here on, by key we mean the primary key $\{1, \ldots, k\}$.

Let R be a relation name with signature $[n, k]$. An R-atom takes the form $R(\underline{s_1, \ldots, s_k}, s_{k+1}, \ldots, s_n)$, where each s_i is a variable or a constant. An R-fact is an R-atom without variables. Two facts are key-equal if they agree on their relation names and on all key positions. For example, $S(\underline{a, b}, d)$ and $S(\underline{a, b}, e)$ are key-equal.

A database (instance) **db** is a finite set of facts. A block in **db** is a maximal set of key-equal facts. Blocks will be separated by dashed lines. A relation in **db** is a maximal set of facts with the same relation name. A database **db** is consistent if it contains no block with two or more facts. A repair of **db** selects exactly one fact from each block.

Example

For the relation name S of signature $[3, 2]$,

the relation

$$
\begin{array}{c|ccc}
S & \underline{1} & \underline{2} & 3 \\
\hline
 & a & b & d \\
 & a & b & e \\
\hdashline
 & b & c & d \\
 & b & c & e \\
 & b & c & f
\end{array}
$$

contains two blocks, and has 6 repairs.

☞ For primary keys, $\oplus$-repairs and C-repairs coincide.

Preliminaries (continued)

A Boolean conjunctive query q is a set of atoms. Satisfaction is defined as usual. If some variable x in q is intended to be free, we write $q(x)$.

Example

$q = \{R(\underline{x}, y), S(\underline{y}, d)\}$, where d is a constant, is satisfied by every database that satisfies the first-order sentence

$$\exists x \exists y \left(R(\underline{x}, y) \wedge S(\underline{y}, d) \right) .$$

$q(x)$ denotes $\exists y \left(R(\underline{x}, y) \wedge S(\underline{y}, d) \right)$.

A conjunctive query is self-join-free if no relation name occurs more than once in it.

CERTAINTY(q)

For every Boolean query q, CERTAINTY(q) is the following decision problem:

Problem CERTAINTY(q)	
Input:	A database **db**.
Question:	Is q true in every repair of **db**?

Note:

- CERTAINTY(q) is a shorthand for $\oplus$-CQA(Σ, q) with Σ the set of key dependencies associated with the relation names in q.
- We study the data complexity, as q is not part of the input.
- If the answer to the question is "yes", then the input is called a "yes"-instance of CERTAINTY(q); otherwise it is a "no"-instance.
- In the remainder, we restrict q to be a self-join-free conjunctive query.

CERTAINTY(q)

For every Boolean query q, CERTAINTY(q) is the following decision problem:

Problem CERTAINTY(q)	
Input:	A database **db**.
Question:	Is q true in every repair of **db**?

Note:

- CERTAINTY(q) is a shorthand for $\oplus$-CQA(Σ, q) with Σ the set of key dependencies associated with the relation names in q.
- We study the data complexity, as q is not part of the input.
- If the answer to the question is "yes", then the input is called a "yes"-instance of CERTAINTY(q); otherwise it is a "no"-instance.
- In the remainder, we restrict q to be a self-join-free conjunctive query.

CERTAINTY(q)

For every Boolean query q, CERTAINTY(q) is the following decision problem:

Problem CERTAINTY(q)	
Input:	A database **db**.
Question:	Is q true in every repair of **db**?

Note:

- CERTAINTY(q) is a shorthand for $\oplus$-CQA(Σ, q) with Σ the set of key dependencies associated with the relation names in q.
- We study the data complexity, as q is not part of the input.
- If the answer to the question is "yes", then the input is called a "yes"-instance of CERTAINTY(q); otherwise it is a "no"-instance.
- In the remainder, we restrict q to be a self-join-free conjunctive query.

CERTAINTY(q)

For every Boolean query q, CERTAINTY(q) is the following decision problem:

Problem CERTAINTY(q)	
Input:	A database **db**.
Question:	Is q true in every repair of **db**?

Note:

- CERTAINTY(q) is a shorthand for $\oplus$-CQA(Σ, q) with Σ the set of key dependencies associated with the relation names in q.
- We study the data complexity, as q is not part of the input.
- If the answer to the question is "yes", then the input is called a "yes"-instance of CERTAINTY(q); otherwise it is a "no"-instance.
- In the remainder, we restrict q to be a self-join-free conjunctive query.

Simple Example

Proposition

For $q = \{S(\underline{y}, d)\}$, CERTAINTY$(q)$ *is in* **FO** *(with d a constant).*

Proof.

The following are equivalent for every database **db**:

- q is true in every repair of **db**; and
- **db** satisfies $\varphi = \exists y \left(\exists u\, S(\underline{y}, u) \wedge \forall u \left(S(\underline{y}, u) \rightarrow u = d\right)\right)$.

□

Example

The following database falsifies φ and therefore is a "no"-instance of CERTAINTY(q).
The repair indicated by $*$ falsifies q.

S	$\underline{1}$	2	
	a	d	
	a	e	$*$
	b	d	
	b	f	$*$

Simple Example

Proposition

For $q = \{S(\underline{y}, d)\}$, CERTAINTY$(q)$ *is in* **FO** *(with* d *a constant).*

Proof.

The following are equivalent for every database **db**:

- q is true in every repair of **db**; and
- **db** satisfies $\varphi = \exists y \left(\exists u\, S(\underline{y}, u) \land \forall u \left(S(\underline{y}, u) \rightarrow u = d\right)\right)$.

□

Example

The following database falsifies φ and therefore is a "no"-instance of CERTAINTY(q).
The repair indicated by $*$ falsifies q.

S	$\underline{1}$	2	
	a	d	
	a	e	$*$
	b	d	
	b	f	$*$

More Involved Example

Proposition

For $q = \{P(\underline{x}, z), N(\underline{y}, z)\}$, CERTAINTY($q$) *is* **coNP**-*complete.*

Proof sketch.

Reduction from MONOTONE SAT. Membership in **coNP** is easy. □

Example

Let $\phi = \overbrace{(p \lor q)}^{1} \land \overbrace{(\neg p \lor \neg z)}^{2} \land \overbrace{(\neg q \lor \neg z)}^{3} \land \overbrace{(z)}^{4}$.

db =

P	$\underline{x}$	z
	1	p
	1	q
	4	z

N	$\underline{y}$	z
	2	p
	2	z
	3	q
	3	z

ϕ is satisfiable

$\Updownarrow$

db has a repair that falsifies $\{P(\underline{x}, z), N(\underline{y}, z)\}$

Complexity Classification Task

Complexity Classification Task

Input: A self-join-free Boolean conjunctive query q.

Task: Determine lower and upper complexity bounds on the complexity of CERTAINTY(q), in terms of common complexity classes like **FO**, **L**, **NL**, **P**, **coNP**.

Example

Input q	Complexity of CERTAINTY(q)
$\{S(\underline{y}, d)\}$	**FO**
$\{P(\underline{x}, z), N(\underline{y}, z)\}$	**coNP**-complete
$\vdots$	$\vdots$

Can we complete this table for every self-join-free Boolean conjunctive query?

Definition (Consistent rewriting)

Let $\mathcal{L}$ be some logic, and $q(\underbrace{x_1, \ldots, x_n}_{\text{free variables}})$ a conjunctive query.

A consistent rewriting in $\mathcal{L}$ of $q(x_1, \ldots, x_n)$ w.r.t. primary keys is a formula $\varphi(x_1, \ldots, x_n)$ in $\mathcal{L}$ such that for every database **db**, the following are equivalent for all constants $c_1, \ldots, c_n$:

1. $q(c_1, \ldots, c_n)$ is true in every repair of **db**; and
2. $\varphi(c_1, \ldots, c_n)$ is true in **db**.

- Gray-colored text is often implicitly understood.
- A rewriting in first-order logic, also called a first-order rewriting, can be expressed in SQL.
- For a Boolean query q,

  q has a first-order rewriting $\stackrel{\text{def.}}{\Longleftrightarrow}$ CERTAINTY(q) is in **FO**.

Descriptive Complexity

Definition (Consistent rewriting)

Let $\mathcal{L}$ be some logic, and $q(\underbrace{x_1, \ldots, x_n}_{\text{free variables}})$ a conjunctive query.

A consistent rewriting in $\mathcal{L}$ of $q(x_1, \ldots, x_n)$ w.r.t. primary keys is a formula $\varphi(x_1, \ldots, x_n)$ in $\mathcal{L}$ such that for every database **db**, the following are equivalent for all constants $c_1, \ldots, c_n$:

1. $q(c_1, \ldots, c_n)$ is true in every repair of **db**; and
2. $\varphi(c_1, \ldots, c_n)$ is true in **db**.

- Gray-colored text is often implicitly understood.
- A rewriting in first-order logic, also called a first-order rewriting, can be expressed in SQL.
- For a Boolean query q,

  q has a first-order rewriting $\overset{\text{def.}}{\Longleftrightarrow}$ CERTAINTY(q) is in **FO**.

Let q be a self-join-free Boolean conjunctive query with an atom $R(\underline{c}, d, y, y, z)$, where c and d are constants.

A database may contain

R	$\underline{1/c}$	$2/d$	$3/y$	$4/y$	$5/z$
	c	d	a_1	a_1	b_1
	c	d	a_2	a_2	b_2
	c	$\boxed{}$	a_3	$\boxed{}$	b_3

.

Let $q' = q \setminus \{R(\underline{c}, d, y, y, z)\}$.

The following can be seen to be equivalent for every database **db**:

1. q is true in every repair of **db**; and
2. **db** satisfies $\exists u_2 \exists u_3 \exists u_4 \exists u_5\, R(\underline{c}, u_2, u_3, u_4, u_5)$ as well as

$$\forall u_2 \forall y \forall u_4 \forall z \left[R(\underline{c}, u_2, y, u_4, z) \rightarrow \begin{pmatrix} (u_2 = d) \wedge (u_4 = y) \wedge \\ \varphi(y, z) \end{pmatrix} \right],$$

where $\varphi(y, z)$ is a rewriting of $q'(y, z)$.

First-Order Rewriting for Variable-Free Keys

Let q be a self-join-free Boolean conjunctive query with an atom $R(\underline{c}, d, y, y, z)$, where c and d are constants.

A database may contain

R	$\underline{1/c}$	$2/d$	$3/y$	$4/y$	$5/z$
	c	d	a_1	a_1	b_1
	c	d	a_2	a_2	b_2
	c	$\boxed{}$	a_3	$\boxed{}$	b_3

.

Let $q' = q \setminus \{R(\underline{c}, d, y, y, z)\}$.

The following can be seen to be equivalent for every database **db**:

1. q is true in every repair of **db**; and
2. **db** satisfies $\exists u_2 \exists u_3 \exists u_4 \exists u_5\ R(\underline{c}, u_2, u_3, u_4, u_5)$ as well as

$$\forall u_2 \forall y \forall u_4 \forall z \left[R(\underline{c}, u_2, y, u_4, z) \rightarrow \begin{pmatrix} (u_2 = d) \wedge (u_4 = y) \wedge \\ \varphi(y, z) \end{pmatrix} \right],$$

where $\varphi(y, z)$ is a rewriting of $q'(y, z)$.

Reifiable Atoms

Definition (Reifiable variable)

Let q be a self-join-free Boolean conjunctive query. We say that a variable x in q is reifiable if for every "yes"-instance $\mathbf{db}$ of CERTAINTY(q),

> *there is a constant c (which depends on $\mathbf{db}$) such that $q_{x\to c}$ is true in every repair of $\mathbf{db}$;*

otherwise x is non-reifiable.

Definition (Reifiable atom)

An atom is reifiable if each variable in its primary key is reifiable.

Example

Let $q = \{R(\underline{x}, y), S(\underline{y}, d)\}$. It can be argued that x is reifiable, but y is not.

Reifiable Atoms

Definition (Reifiable variable)

Let q be a self-join-free Boolean conjunctive query. We say that a variable x in q is reifiable if for every "yes"-instance **db** of CERTAINTY(q),

> *there is a constant c (which depends on* **db***) such that $q_{x \to c}$ is true in every repair of* **db***;*

otherwise x is non-reifiable.

Definition (Reifiable atom)

An atom is reifiable if each variable in its primary key is reifiable.

Example

Let $q = \{R(\underline{x}, y), S(\underline{y}, d)\}$. It can be argued that x is reifiable, but y is not.

Example (continued)

y is not reifiable in $\{R(\underline{x}, y), S(\underline{y}, d)\}$, as shown by:

R	$\underline{x}$	y
	a	c_1
	a	c_2

S	$\underline{y}$	d
	c_1	d
	c_2	d

There are two repairs, both satisfying q. However, there is no constant c such that $q_{y \to c}$ is true in every repair.

First-Order Rewriting for Reifiable Atoms

Let q be a self-join-free Boolean conjunctive query with an atom $R(\underline{c, y, y, z}, \ldots)$ such that y and z are reifiable.

Then, the following are equivalent for every database **db**:

1. q is true in every repair of **db**; and
2. **db** satisfies

$$\underbrace{\exists y \exists z}_{\text{"reify"}} \; \psi(y, z),$$

where $\psi(y, z)$ is a rewriting of $q(y, z)$.

Crux: In $q(y, z)$, the key of $R(\underline{c, y, y, z}, \ldots)$ contains only constants and free variables, and we can apply the rewriting seen previously for variable-free keys.

Claim: In a self-join-free conjunctive query, free variables can be treated as constants.

First-Order Rewriting for Reifiable Atoms

Let q be a self-join-free Boolean conjunctive query with an atom $R(\underline{c, y, y, z}, \ldots)$ such that y and z are reifiable.

Then, the following are equivalent for every database **db**:

1. q is true in every repair of **db**; and
2. **db** satisfies

$$\underbrace{\exists y \exists z}_{\text{“reify”}} \ \psi(y, z),$$

where $\psi(y, z)$ is a rewriting of $q(y, z)$.

Crux: In $q(y, z)$, the key of $R(\underline{c, y, y, z}, \ldots)$ contains only constants and free variables, and we can apply the rewriting seen previously for variable-free keys.

Claim: In a self-join-free conjunctive query, free variables can be treated as constants.

First-Order Rewriting for Reifiable Atoms

Let q be a self-join-free Boolean conjunctive query with an atom $R(\underline{c, y, y, z}, \ldots)$ such that y and z are reifiable.

Then, the following are equivalent for every database **db**:

1. q is true in every repair of **db**; and
2. **db** satisfies

$$\underbrace{\exists y \exists z}_{\text{“reify”}} \ \psi(y, z),$$

where $\psi(y, z)$ is a rewriting of $q(y, z)$.

Crux: In $q(y, z)$, the key of $R(\underline{c, y, y, z}, \ldots)$ contains only constants and free variables, and we can apply the rewriting seen previously for variable-free keys.

Claim: In a self-join-free conjunctive query, free variables can be treated as constants.

Example

A first-order rewriting of $q = \{R(\underline{x}, y), S(\underline{y}, d)\}$ is constructed as follows:

1. since x is reifiable, a rewriting of q is $\exists x\ \psi(x)$, where $\psi(x)$ is a rewriting of $q(x)$;
2. by treating the free variable x in $q(x)$ as a constant,
$$\psi(x) = \exists u\ R(\underline{x}, u) \wedge \forall y\, (R(\underline{x}, y) \rightarrow \varphi(x, y))\,,$$
where $\varphi(x, y)$ is a rewriting of $q'(y) = S(\underline{y}, d)$;
3. Finally, $\varphi(x, y) = \exists u\ S(\underline{y}, u) \wedge \forall u\, (S(\underline{y}, u) \rightarrow u = d)$.

Putting everything together:

$$\exists x \left(\begin{array}{l} \exists u\ R(\underline{x}, u) \wedge \\ \forall y \left(R(\underline{x}, y) \rightarrow \left(\begin{array}{l} \exists u\ S(\underline{y}, u) \wedge \\ \forall u\, (S(\underline{y}, u) \rightarrow u = d) \end{array} \right) \right) \end{array} \right).$$

Coming next: Can we determine whether a variable is reifiable?

Example

A first-order rewriting of $q = \{R(\underline{x}, y), S(\underline{y}, d)\}$ is constructed as follows:

1. since x is reifiable, a rewriting of q is $\exists x\, \psi(x)$, where $\psi(x)$ is a rewriting of $q(x)$;
2. by treating the free variable x in $q(x)$ as a constant,
$$\psi(x) = \exists u\, R(\underline{x}, u) \wedge \forall y\, (R(\underline{x}, y) \rightarrow \varphi(x, y)),$$
where $\varphi(x, y)$ is a rewriting of $q'(y) = S(\underline{y}, d)$;
3. Finally, $\varphi(x, y) = \exists u\, S(\underline{y}, u) \wedge \forall u\, (S(\underline{y}, u) \rightarrow u = d)$.

Putting everything together:

$$\exists x \left(\begin{array}{l} \exists u\, R(\underline{x}, u) \wedge \\ \forall y \left(R(\underline{x}, y) \rightarrow \left(\begin{array}{l} \exists u\, S(\underline{y}, u) \wedge \\ \forall u\, (S(\underline{y}, u) \rightarrow u = d) \end{array} \right) \right) \end{array} \right).$$

Coming next: Can we determine whether a variable is reifiable?

Example

A first-order rewriting of $q = \{R(\underline{x}, y), S(\underline{y}, d)\}$ is constructed as follows:

1. since x is reifiable, a rewriting of q is $\exists x\, \psi(x)$, where $\psi(x)$ is a rewriting of $q(x)$;
2. by treating the free variable x in $q(x)$ as a constant,
$$\psi(x) = \exists u\, R(\underline{x}, u) \wedge \forall y\, (R(\underline{x}, y) \rightarrow \varphi(x, y)),$$
where $\varphi(x, y)$ is a rewriting of $q'(y) = S(\underline{y}, d)$;
3. Finally, $\varphi(x, y) = \exists u\, S(\underline{y}, u) \wedge \forall u\, \left(S(\underline{y}, u) \rightarrow u = d\right)$.

Putting everything together:

$$\exists x \left(\begin{array}{l} \exists u\, R(\underline{x}, u) \wedge \\ \forall y \left(R(\underline{x}, y) \rightarrow \left(\begin{array}{l} \exists u\, S(\underline{y}, u) \wedge \\ \forall u\, (S(\underline{y}, u) \rightarrow u = d) \end{array} \right) \right) \end{array} \right).$$

Coming next: Can we determine whether a variable is reifiable?

Example

A first-order rewriting of $q = \{R(\underline{x}, y), S(\underline{y}, d)\}$ is constructed as follows:

1. since x is reifiable, a rewriting of q is $\exists x\, \psi(x)$, where $\psi(x)$ is a rewriting of $q(x)$;
2. by treating the free variable x in $q(x)$ as a constant,
$$\psi(x) = \exists u\, R(\underline{x}, u) \wedge \forall y\, (R(\underline{x}, y) \rightarrow \varphi(x, y)),$$
where $\varphi(x, y)$ is a rewriting of $q'(y) = S(\underline{y}, d)$;
3. Finally, $\varphi(x, y) = \exists u\, S(\underline{y}, u) \wedge \forall u\, (S(\underline{y}, u) \rightarrow u = d)$.

Putting everything together:

$$\exists x \left(\begin{array}{l} \exists u\, R(\underline{x}, u) \wedge \\ \forall y \left(R(\underline{x}, y) \rightarrow \left(\begin{array}{l} \exists u\, S(\underline{y}, u) \wedge \\ \forall u\, (S(\underline{y}, u) \rightarrow u = d) \end{array} \right) \right) \end{array} \right).$$

Coming next: Can we determine whether a variable is reifiable?

Attacks by Example

Let $q = \{R(\underline{u}, x), S(\underline{x}, y), T(\underline{y}, z)\}$. The following database shows that x, y, and z are not reifiable.

R	$\underline{u}$	x
	a	c_1
	a	c_2

S	$\underline{x}$	y
	c_1	d_1
	c_2	d_2

T	$\underline{y}$	z
	d_1	e_1
	d_2	e_2

Only the R-relation is inconsistent, yielding two repairs, both satisfying q (i.e., it is a "yes"-instance). But

- there is no c such that both repairs satisfy $q_{x \to c}$;
- there is no d such that both repairs satisfy $q_{y \to d}$; and
- there is no e such that both repairs satisfy $q_{z \to e}$.

We will write $R \overset{q}{\rightsquigarrow} x$, $R \overset{q}{\rightsquigarrow} y$, and $R \overset{q}{\rightsquigarrow} z$.

Also $R \overset{q}{\rightsquigarrow} S$, and $R \overset{q}{\rightsquigarrow} T$.

$\overset{q}{\rightsquigarrow}$ is read "attacks" and is used to show non-reifiability.

Attacks by Example

Let $q = \{R(\underline{u}, x), S(\underline{x}, y), T(\underline{y}, z)\}$. The following database shows that x, y, and z are not reifiable.

R	$\underline{u}$	x
	a	c_1
	a	c_2

S	$\underline{x}$	y
	c_1	d_1
	c_2	d_2

T	$\underline{y}$	z
	d_1	e_1
	d_2	e_2

Only the R-relation is inconsistent, yielding two repairs, both satisfying q (i.e., it is a "yes"-instance). But

- there is no c such that both repairs satisfy $q_{x \to c}$;
- there is no d such that both repairs satisfy $q_{y \to d}$; and
- there is no e such that both repairs satisfy $q_{z \to e}$.

We will write $R \overset{q}{\rightsquigarrow} x$, $R \overset{q}{\rightsquigarrow} y$, and $R \overset{q}{\rightsquigarrow} z$.

Also $R \overset{q}{\rightsquigarrow} S$, and $R \overset{q}{\rightsquigarrow} T$.

$\overset{q}{\rightsquigarrow}$ is read "attacks" and is used to show non-reifiability.

Attacks by Example

Let $q = \{R(\underline{u}, x), S(\underline{x}, y), T(\underline{y}, z)\}$. The following database shows that x, y, and z are not reifiable.

R	$\underline{u}$	x
	a	c_1
	a	c_2

S	$\underline{x}$	y
	c_1	d_1
	c_2	d_2

T	$\underline{y}$	z
	d_1	e_1
	d_2	e_2

Only the R-relation is inconsistent, yielding two repairs, both satisfying q (i.e., it is a "yes"-instance). But

- there is no c such that both repairs satisfy $q_{x \to c}$;
- there is no d such that both repairs satisfy $q_{y \to d}$; and
- there is no e such that both repairs satisfy $q_{z \to e}$.

We will write $R \overset{q}{\rightsquigarrow} x$, $R \overset{q}{\rightsquigarrow} y$, and $R \overset{q}{\rightsquigarrow} z$.

Also $R \overset{q}{\rightsquigarrow} S$, and $R \overset{q}{\rightsquigarrow} T$.

$\overset{q}{\rightsquigarrow}$ is read "attacks" and is used to show non-reifiability.

Attacks by Example 2 (Continued)

Let $q = \{R(\underline{u}, x), R(\underline{x}, y), T(\underline{y}, z)\}$. The following database shows that y and z are not reifiable.

R	$\underline{u}$	x
	a	c

S	$\underline{x}$	y
	c	d_1
	c	d_2

T	$\underline{y}$	z
	d_1	e_1
	d_2	e_2

Only the S-relation is inconsistent, yielding two repairs, both satisfying q. But,

- there is no d such that both repairs satisfy $q_{y \to d}$; and
- there is no e such that both repairs satisfy $q_{z \to e}$.

We have $S \overset{q}{\rightsquigarrow} y$ and $S \overset{q}{\rightsquigarrow} z$.

Also $S \overset{q}{\rightsquigarrow} T$ (because S attacks a key-variable of T).

Attacks by Example 2 (Continued)

Let $q = \{R(\underline{u}, x), R(\underline{x}, y), T(\underline{y}, z)\}$. The following database shows that y and z are not reifiable.

R	$\underline{u}$	x
	a	c

S	$\underline{x}$	y
	c	d_1
	c	d_2

T	$\underline{y}$	z
	d_1	e_1
	d_2	e_2

Only the S-relation is inconsistent, yielding two repairs, both satisfying q. But,

- there is no d such that both repairs satisfy $q_{y \to d}$; and
- there is no e such that both repairs satisfy $q_{z \to e}$.

We have $S \overset{q}{\rightsquigarrow} y$ and $S \overset{q}{\rightsquigarrow} z$.

Also $S \overset{q}{\rightsquigarrow} T$ (because S attacks a key-variable of T).

Attacks by Example 2 (Continued)

Let $q = \{R(\underline{u}, x), R(\underline{x}, y), T(\underline{y}, z)\}$. The following database shows that y and z are not reifiable.

R	$\underline{u}$	x
	a	c

S	$\underline{x}$	y
	c	d_1
	c	d_2

T	$\underline{y}$	z
	d_1	e_1
	d_2	e_2

Only the S-relation is inconsistent, yielding two repairs, both satisfying q. But,

- there is no d such that both repairs satisfy $q_{y \to d}$; and
- there is no e such that both repairs satisfy $q_{z \to e}$.

We have $S \overset{q}{\rightsquigarrow} y$ and $S \overset{q}{\rightsquigarrow} z$.

Also $S \overset{q}{\rightsquigarrow} T$ (because S attacks a key-variable of T).

Let $q = \{P(\underline{u}, y), R(\underline{u}, x), S(\underline{x}, y), T(\underline{y}, z)\}$.

Claim: $R \overset{q}{\not\rightsquigarrow} y$ and $R \overset{q}{\not\rightsquigarrow} z$.

Note that the following is a "no"-instance because the repair indicated by $*$ falsifies q:

P	$\underline{u}$	y	
	a	d_1	$*$
	a	d_2	

R	$\underline{u}$	x	
	a	c_1	
	a	c_2	$*$

S	$\underline{x}$	y
	c_1	d_1
	c_2	d_2

T	$\underline{y}$	z
	d_1	e_1
	d_2	e_2

By definition, to show that variables are non-reifiable, we need "yes"-instances!!!

Attacks by Example 4 (Continued)

Let $q = \{P(\underline{u}, y), R(\underline{u}, x), S(\underline{x}, y), T(\underline{y}, z)\}$.

Claim: $R \overset{q}{\rightsquigarrow} x$.

The following database has two repairs, both satisfying q, for different valuations of x:

P	$\underline{u}$	y
	a	d

R	$\underline{u}$	x
	a	c_1
	a	c_2

S	$\underline{x}$	y
	c_1	d
	c_2	d

T	$\underline{y}$	z
	d	e

Syntactic Characterization of Attacks

Let q be a self-join-free Boolean conjunctive query containing an atom with relation name R.
By an abuse of terminology, this atom is also referred to as the atom R (of q). This is well-defined, as q is self-join-free.

We write $\mathrm{vars}(R)$ for the set of variables occurring in R, and $\mathrm{key}(R)$ for the set of variables occurring in the key of R.

We define $\mathcal{K}(q)$ as the set of functional dependencies that contains $\mathrm{key}(R) \rightarrow \mathrm{vars}(R)$ for every atom R of q.

We define $R^{+,q} = \{x \mid \mathcal{K}(q \setminus \{R\}) \models \mathrm{key}(R) \rightarrow x\}$, the set of variables (in q) that are externally determined by (the key of) R.

Example

If $q = \{P(\underline{u}, y), R(\underline{u}, x), S(\underline{x}, y), T(\underline{y}, z)\}$,
then $\mathcal{K}(q) = \{u \rightarrow y, u \rightarrow x, x \rightarrow y, y \rightarrow z\}$ and $R^{+,q} = \{u, y, z\}$.

Syntactic Characterization of Attacks

Let q be a self-join-free Boolean conjunctive query containing an atom with relation name R.
By an abuse of terminology, this atom is also referred to as the atom R (of q). This is well-defined, as q is self-join-free.

We write $\mathsf{vars}(R)$ for the set of variables occurring in R, and $\mathsf{key}(R)$ for the set of variables occurring in the key of R.

We define $\mathcal{K}(q)$ as the set of functional dependencies that contains $\mathsf{key}(R) \rightarrow \mathsf{vars}(R)$ for every atom R of q.

We define $R^{+,q} = \{x \mid \mathcal{K}(q \setminus \{R\}) \models \mathsf{key}(R) \rightarrow x\}$, the set of variables (in q) that are externally determined by (the key of) R.

Example

If $q = \{P(\underline{u}, y), R(\underline{u}, x), S(\underline{x}, y), T(\underline{y}, z)\}$,
then $\mathcal{K}(q) = \{u \rightarrow y, u \rightarrow x, x \rightarrow y, y \rightarrow z\}$ and $R^{+,q} = \{u, y, z\}$.

Syntactic Characterization of Attacks

Let q be a self-join-free Boolean conjunctive query containing an atom with relation name R.
By an abuse of terminology, this atom is also referred to as the atom R (of q). This is well-defined, as q is self-join-free.

We write $\mathsf{vars}(R)$ for the set of variables occurring in R, and $\mathsf{key}(R)$ for the set of variables occurring in the key of R.

We define $\mathcal{K}(q)$ as the set of functional dependencies that contains $\mathsf{key}(R) \rightarrow \mathsf{vars}(R)$ for every atom R of q.

We define $R^{+,q} = \{x \mid \mathcal{K}(q \setminus \{R\}) \models \mathsf{key}(R) \rightarrow x\}$, the set of variables (in q) that are externally determined by (the key of) R.

Example

If $q = \{P(\underline{u}, y), R(\underline{u}, x), S(\underline{x}, y), T(\underline{y}, z)\}$,
then $\mathcal{K}(q) = \{u \rightarrow y, u \rightarrow x, x \rightarrow y, y \rightarrow z\}$ and $R^{+,q} = \{u, y, z\}$.

Syntactic Characterization of Attacks (continued)

Definition ($\overset{q}{\rightsquigarrow}$)

Let q be a self-join-free Boolean conjunctive query. We write $R \overset{q}{\rightsquigarrow} z$ if there exists a sequence of variables

$$x_1, x_2, \ldots, x_n$$

such that

1. $x_1 \in \text{vars}(R)$ and $x_n = z$;
2. no x_i is externally determined by R; and
3. every two adjacent variables occur together in some atom of q.

Example

- If $q = \{R(\underline{u}, x), S(\underline{x}, y), T(\underline{y}, z)\}$, then the sequence x, y, z shows that $R \overset{q}{\rightsquigarrow} z$ (note here that $R^{+,q} = \{u\}$).
- When we add $P(\underline{u}, y)$, then y (as well as z) becomes externally determined by R, making the attack disappear.

Syntactic Characterization of Attacks (continued)

Definition ($\overset{q}{\rightsquigarrow}$)

Let q be a self-join-free Boolean conjunctive query. We write $R \overset{q}{\rightsquigarrow} z$ if there exists a sequence of variables

$$x_1, x_2, \ldots, x_n$$

such that

1. $x_1 \in \text{vars}(R)$ and $x_n = z$;
2. no x_i is externally determined by R; and
3. every two adjacent variables occur together in some atom of q.

Example

- If $q = \{R(\underline{u}, x), S(\underline{x}, y), T(\underline{y}, z)\}$, then the sequence x, y, z shows that $R \overset{q}{\rightsquigarrow} z$ (note here that $R^{+,q} = \{u\}$).
- When we add $P(\underline{u}, y)$, then y (as well as z) becomes externally determined by R, making the attack disappear.

Attack Graph

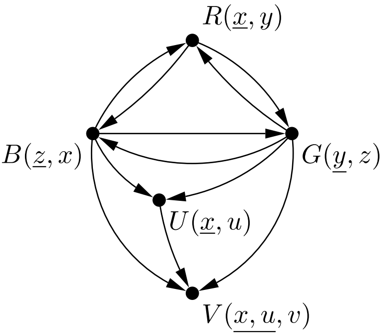
Definition (Attack graph)

Let q be a self-join-free Boolean conjunctive query.

We write $R \overset{q}{\rightsquigarrow} S$ $(R \neq S)$ if $R \overset{q}{\rightsquigarrow} z$ for some z in key(S).

The attack graph of q is a directed graph whose vertices are the atoms of q; there is a directed edge from R to S if $R \overset{q}{\rightsquigarrow} S$.

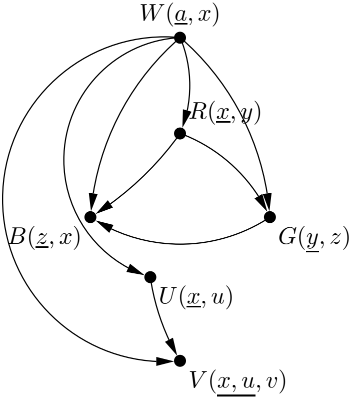
Examples of Attack Graph I



$$
\begin{aligned}
R^{+,q} &= \{x, u, v\} \\
B^{+,q} &= \{z\} \\
G^{+,q} &= \{y\} \\
U^{+,q} &= \{x, y, z\} \\
V^{+,q} &= \{x, u, y, z\}
\end{aligned}
$$

- $R \overset{q}{\rightsquigarrow} B$ because of the sequence y, z.
- $G \overset{q}{\rightsquigarrow} V$ because of the sequence z, x.
- $U \overset{q}{\rightsquigarrow} V$ because of the sequence u.
- Etc.

Examples of Attack Graph II



Note that $\mathcal{K}(q)$ contains $\emptyset \rightarrow x$ because of W.

$$
\begin{aligned}
W^{+,q} &= \{\} \\
R^{+,q} &= \{x, u, v\} \\
B^{+,q} &= \{z, x, y, u, v\} \\
G^{+,q} &= \{y, x, u, v\} \\
U^{+,q} &= \{x, y, z\} \\
V^{+,q} &= \{x, u, y, z\}
\end{aligned}
$$

- $R \overset{q}{\rightsquigarrow} B$ because of the sequence y, z.
- $G \overset{q}{\not\rightsquigarrow} V$ because $\{x, u\} \subseteq G^{+,q}$.
- Etc.

Finale

Theorem

Let q be a self-join-free Boolean conjunctive query. If the attack graph of q is acyclic, then q has a consistent first-order rewriting.

Proof sketch.

Assume that q's attack graph is acyclic. Then q must contain an atom R that is unattacked, and therefore reifiable. We can rewrite this atom by the rewriting seen previously for reifiable atoms. It can be shown that the attack graph of $q \setminus \{R\}$ remains acyclic (where it is understood that the variables of $\text{vars}(R)$ are treated as constants in $q \setminus \{R\}$). □

The converse is also true:

Theorem

If the attack graph of q is cyclic, then q has no consistent first-order rewriting.

Finale

Theorem

Let q be a self-join-free Boolean conjunctive query. If the attack graph of q is acyclic, then q has a consistent first-order rewriting.

Proof sketch.

Assume that q's attack graph is acyclic. Then q must contain an atom R that is unattacked, and therefore reifiable. We can rewrite this atom by the rewriting seen previously for reifiable atoms. It can be shown that the attack graph of $q \setminus \{R\}$ remains acyclic (where it is understood that the variables of $\mathrm{vars}(R)$ are treated as constants in $q \setminus \{R\}$). □

The converse is also true:

Theorem

If the attack graph of q is cyclic, then q has no consistent first-order rewriting.

Finale

Theorem

Let q be a self-join-free Boolean conjunctive query. If the attack graph of q is acyclic, then q has a consistent first-order rewriting.

Proof sketch.

Assume that q's attack graph is acyclic. Then q must contain an atom R that is unattacked, and therefore reifiable. We can rewrite this atom by the rewriting seen previously for reifiable atoms. It can be shown that the attack graph of $q \setminus \{R\}$ remains acyclic (where it is understood that the variables of $\mathrm{vars}(R)$ are treated as constants in $q \setminus \{R\}$). $\square$

The converse is also true:

Theorem

If the attack graph of q is cyclic, then q has no consistent first-order rewriting.

Cyclic Attack Graphs

Recall (by our reduction from MONOTONE SAT):

Proposition

For $q = \{P(\underline{x}, z), N(\underline{y}, z)\}$, CERTAINTY$(q)$ is **coNP**-*complete.*

One can prove:

Proposition

For $q = \{R(\underline{x}, y), S(\underline{y}, x)\}$, CERTAINTY$(q)$ is in **P** *(but not in* **FO**, *because of the cycle $R \overset{q}{\rightsquigarrow} S \overset{q}{\rightsquigarrow} R$).*

Definition (Weak attack)

Let q be a self-join-free Boolean conjunctive query. An attack $R \overset{q}{\rightsquigarrow} S$ is weak if $\mathcal{K}(q) \models \mathrm{key}(R) \rightarrow \mathrm{key}(S)$; otherwise it is *strong*.

Cyclic Attack Graphs

Recall (by our reduction from MONOTONE SAT):

Proposition

For $q = \{P(\underline{x}, z), N(\underline{y}, z)\}$, CERTAINTY$(q)$ *is* **coNP**-*complete.*

One can prove:

Proposition

For $q = \{R(\underline{x}, y), S(\underline{y}, x)\}$, CERTAINTY$(q)$ *is in* **P** *(but not in* **FO**, *because of the cycle* $R \overset{q}{\rightsquigarrow} S \overset{q}{\rightsquigarrow} R$*).*

Definition (Weak attack)

Let q be a self-join-free Boolean conjunctive query. An attack $R \overset{q}{\rightsquigarrow} S$ is weak if $\mathcal{K}(q) \models \text{key}(R) \rightarrow \text{key}(S)$; otherwise it is *strong*.

Trichotomy Theorem

Theorem ([KW21])

Let q be a self-join-free Boolean conjunctive query.

- *If the attack graph of q is acyclic, then* CERTAINTY(q) *is in* **FO**;
- *if the attack graph of q is cyclic but no cycle contains a strong attack, then* CERTAINTY(q) *is* **L**-*complete;*
- *otherwise* CERTAINTY(q) *is* **coNP**-*complete.*

Dichotomy

Corollary

For every self-join-free Boolean conjunctive query q, coCERTAINTY(q) *is either in* **P** *or* **NP***-complete.*

Recall from [GJ79]:

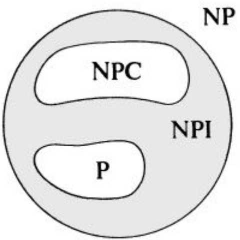


Figure 7.1 The world of NP, reprised (assuming P $\neq$ NP).

Does it Extend to (Unions of) Conjunctive Queries???

Conjecture

For every Boolean conjunctive query q, CERTAINTY(q) *is either in* **P** *or* **coNP**-*complete.*

Why Is It Hard to Obtain a Dichotomy for Consistent Query Answering?

In ACM TOCL, 2015

GAËLLE FONTAINE, University of Chile

A database may for various reasons become inconsistent with respect to a given set of integrity constraints. In the late 1990s, the formal approach of consistent query answering was proposed in order to query such databases. Since then, a lot of efforts have been spent to classify the complexity of consistent query answering under various classes of constraints. It is known that for the most common constraints and queries, the problem is in coNP and might be coNP-hard, yet several relevant tractable classes have been identified. Additionally, the results that emerged suggested that given a set of key constraints and a conjunctive query, the problem of consistent query answering is either in PTIME or is coNP-complete. However, despite all the work, as of today this dichotomy remains a conjecture.

The main contribution of this article is to explain why it appears so difficult to obtain a dichotomy result in the setting of consistent query answering. Namely, we prove that such a dichotomy with respect to common classes of constraints and queries is harder to achieve than a dichotomy for the constraint satisfaction problem, which is a famous open problem since the 1990s.

THEOREM 4.1. *There is a key constraint ϕ such that for each structure $\mathbb{B}$, we can compute a Boolean UCQ q using constants such that cHom*($\mathbb{B}$) *and* $\overline{CQA}(q, \phi)$ *are polynomially equivalent.*

As a consequence, a dichotomy result for consistent query answering with respect to keys and UCQs with constants would provide an alternative proof for the dichotomy theorem for conservative CSP.

Adding Foreign Keys [HW22]

Proposition

Let $q = \{N(\underline{x}, c, y), O(\underline{y})\}$ *and* $\mathcal{FK} = \{N[3] \subseteq O[1]\}$. *Then,* $\oplus$-CQA$(q, \mathcal{PK} \cup \mathcal{FK})$ *is not in* **FO** *(because it is not Hanf-local).*

Proof idea.

db =

N

$\underline{x}$	c	y
b_1	c	1
b_1	d	2
b_2	c	2
b_2	d	3
b_3	c	3
b_3	d	4
$\vdots$	$\vdots$	$\vdots$
b_n	c	n
b_n	d	$n+1$
b_{n+1}	$\square$	$n+1$

O

$\underline{y}$
1

Is **db** a "yes"- or a "no"-instance?

Our goal: construct a repair **r** s.t. $\mathbf{r} \not\models q$.

$\square$

Adding Foreign Keys [HW22]

Proposition

Let $q = \{N(\underline{x}, c, y), O(\underline{y})\}$ *and* $\mathcal{FK} = \{N[3] \subseteq O[1]\}$*. Then,* $\oplus$-CQA$(q, \mathcal{PK} \cup \mathcal{FK})$ *is not in* **FO** *(because it is not Hanf-local).*

Proof idea.

db =

N

$\underline{x}$	c	y
b_1	c	1
b_1	d	2
b_2	c	2
b_2	d	3
b_3	c	3
b_3	d	4
$\vdots$	$\vdots$	$\vdots$
b_n	c	n
b_n	d	$n+1$
b_{n+1}	$\Box$	$n+1$

O

	$\underline{y}$
$\circ$	1

Is **db** a "yes"- or a "no"-instance?

Our goal: construct a repair **r** s.t. $\mathbf{r} \not\models q$.

$\Box$

Adding Foreign Keys [HW22]

Proposition

Let $q = \{N(\underline{x}, c, y), O(\underline{y})\}$ *and* $\mathcal{FK} = \{N[3] \subseteq O[1]\}$. *Then,* $\oplus$-CQA$(q, \mathcal{PK} \cup \mathcal{FK})$ *is not in* **FO** *(because it is not Hanf-local).*

Proof idea.

db =

	N: $\underline{x}$	c	y
	b_1	c	1
$\circ$	b_1	d	2
	b_2	c	2
	b_2	d	3
	b_3	c	3
	b_3	d	4
	$\vdots$	$\vdots$	$\vdots$
	b_n	c	n
	b_n	d	$n+1$
	b_{n+1}	$\square$	$n+1$

	O: $\underline{y}$
$\circ$	1

Is **db** a "yes"- or a "no"-instance?

Our goal: construct a repair **r** s.t. $\mathbf{r} \not\models q$.

$\square$

Adding Foreign Keys [HW22]

Proposition

Let $q = \{N(\underline{x}, c, y), O(\underline{y})\}$ *and* $\mathcal{FK} = \{N[3] \subseteq O[1]\}$. *Then,* $\oplus$-CQA$(q, \mathcal{PK} \cup \mathcal{FK})$ *is not in* **FO** *(because it is not Hanf-local).*

Proof idea.

db =

N:

	$\underline{x}$	c	y
	b_1	c	1
$\circ$	b_1	d	2
	b_2	c	2
	b_2	d	3
	b_3	c	3
	b_3	d	4
	$\vdots$	$\vdots$	$\vdots$
	b_n	c	n
	b_n	d	$n+1$
	b_{n+1}	$\square$	$n+1$

O:

	$\underline{y}$
$\circ$	1
$\circ$	2

Is **db** a "yes"- or a "no"-instance?

Our goal: construct a repair **r** s.t. $\mathbf{r} \not\models q$.

$\square$

Adding Foreign Keys [HW22]

Proposition

Let $q = \{N(\underline{x}, c, y), O(\underline{y})\}$ *and* $\mathcal{FK} = \{N[3] \subseteq O[1]\}$*. Then,* $\oplus$-CQA$(q, \mathcal{PK} \cup \mathcal{FK})$ *is not in* **FO** *(because it is not Hanf-local).*

Proof idea.

db =

	N		
	$\underline{x}$	c	y
	b_1	c	1
○	b_1	d	2
	b_2	c	2
○	b_2	d	3
	b_3	c	3
	b_3	d	4
	⋮	⋮	⋮
	b_n	c	n
	b_n	d	$n+1$
	b_{n+1}	$\square$	$n+1$

	O
	$\underline{y}$
○	1
○	2

Is **db** a "yes"- or a "no"-instance?

Our goal: construct a repair **r** s.t. $\mathbf{r} \not\models q$.

$\square$

Proposition

Let $q = \{N(\underline{x}, c, y), O(\underline{y})\}$ *and* $\mathcal{FK} = \{N[3] \subseteq O[1]\}$. *Then,* $\oplus$-CQA$(q, \mathcal{PK} \cup \mathcal{FK})$ *is not in* **FO** *(because it is not Hanf-local).*

Proof idea.

db =

N	$\underline{x}$	c	y
	b_1	c	1
$\circ$	b_1	d	2
	b_2	c	2
$\circ$	b_2	d	3
	b_3	c	3
	b_3	d	4
	$\vdots$	$\vdots$	$\vdots$
	b_n	c	n
	b_n	d	$n+1$
	b_{n+1}	$\square$	$n+1$

O	$\underline{y}$
$\circ$	1
$\circ$	2
$\circ$	3

Is **db** a "yes"- or a "no"-instance?

Our goal: construct a repair **r** s.t. **r** $\not\models q$.

$\square$

Adding Foreign Keys [HW22]

Proposition

Let $q = \{N(\underline{x}, c, y), O(\underline{y})\}$ *and* $\mathcal{FK} = \{N[3] \subseteq O[1]\}$. *Then,* $\oplus$-CQA$(q, \mathcal{PK} \cup \mathcal{FK})$ *is not in* **FO** *(because it is not Hanf-local).*

Proof idea.

db =

	N		
	$\underline{x}$	c	y
	b_1	c	1
$\circ$	b_1	d	2
	b_2	c	2
$\circ$	b_2	d	3
	b_3	c	3
$\circ$	b_3	d	4
	$\vdots$	$\vdots$	$\vdots$
	b_n	c	n
$\circ$	b_n	d	$n+1$
$\circ$	b_{n+1}	$\square$	$n+1$

	O
	$\underline{y}$
$\circ$	1
$\circ$	2
$\circ$	3
$\circ$	4
$\vdots$	$\vdots$
$\circ$	$n+1$

Is **db** a "yes"- or a "no"-instance?

Our goal: construct a repair **r** s.t. $\mathbf{r} \not\models q$.

$\square$

Adding Foreign Keys [HW22]

Proposition

Let $q = \{N(\underline{x}, c, y), O(\underline{y})\}$ *and* $\mathcal{FK} = \{N[3] \subseteq O[1]\}$. *Then,* $\oplus$-CQA$(q, \mathcal{PK} \cup \mathcal{FK})$ *is not in* **FO** *(because it is not Hanf-local).*

Proof idea.

db =

N

	$\underline{x}$	c	y
	b_1	c	1
∘	b_1	d	2
	b_2	c	2
∘	b_2	d	3
	b_3	c	3
∘	b_3	d	4
	⋮	⋮	⋮
	b_n	c	n
∘	b_n	d	$n+1$
∘	b_{n+1}	$\square$	$n+1$

O

	$\underline{y}$
∘	1
∘	2
∘	3
∘	4
⋮	⋮
∘	$n+1$

Is **db** a "yes"- or a "no"-instance?

Our goal: construct a repair **r** s.t. **r** $\not\models q$.

The goal is achievable iff we reach $\square \neq c$.

$\square$

Counting

For every Boolean query q, $\sharp$CERTAINTY(q) is the following problem:

Problem $\sharp$CERTAINTY(q)

Input: A database **db**.

Question: How many repairs of **db** satisfy q?

Complexity Classification Task

Input: A self-join-free Boolean conjunctive query q.

Task: Determine lower and upper complexity bounds on the complexity of $\sharp$CERTAINTY(q), in terms of common complexity classes like **FP** and $\sharp$**P**.

- See [MW13] and its generalization [CLPS22] to functional dependencies.
- Same problem as query answering in block-independent disjoint (BID) probabilistic databases under the restriction that in every block **b**, every tuple has probability $\frac{1}{|\mathbf{b}|}$.

Counting

For every Boolean query q, $\sharp\mathsf{CERTAINTY}(q)$ is the following problem:

Problem $\sharp\mathsf{CERTAINTY}(q)$

Input: A database **db**.

Question: How many repairs of **db** satisfy q?

Complexity Classification Task

Input: A self-join-free Boolean conjunctive query q.

Task: Determine lower and upper complexity bounds on the complexity of $\sharp\mathsf{CERTAINTY}(q)$, in terms of common complexity classes like **FP** and $\sharp$**P**.

- See [MW13] and its generalization [CLPS22] to functional dependencies.
- Same problem as query answering in block-independent disjoint (BID) probabilistic databases under the restriction that in every block $\mathbf{b}$, every tuple has probability $\frac{1}{|\mathbf{b}|}$.

BID Databases

Every input to CERTAINTY(q) is a block-independent disjoint database without probabilities (or with uniform probabilities).

☞ Inconsistency is not only a burden, but also a chance. [3]

Researchers:

	Name	Affiliation	P
t_1^1	Fred	U. Washington	$p_1^1 = 0.3$
t_1^2		U. Wisconsin	$p_1^2 = 0.2$
t_1^3		Y! Research	$p_1^3 = 0.5$
t_2^1	Sue	U. Washington	$p_2^1 = 1.0$
t_3^1	John	U. Wisconsin	$p_3^1 = 0.7$
t_3^2		U. Washington	$p_3^2 = 0.3$
t_4^1	Frank	Y! Research	$p_4^1 = 0.9$
t_4^2		M. Research	$p_4^2 = 0.1$

[3] Inspired by [KL17]. The image is from [DRS09].

Queries with Aggregation [ABC01]

WorksFor

Emp	*Dept*	*Sal*
Ed	Toys	1000
Ed	Shoes	1500
An	Toys	2000
Tim	Shoes	4000

SELECT Dept, SUM(Sal)
FROM WorksFor
GROUP BY Dept

$\mathbf{r}_1$

Emp	*Dept*	*Sal*
Ed	Toys	1000
An	Toys	2000
Tim	Shoes	4000

$\Rightarrow$

Dept	*SUM(Sal)*
Toys	3000
Shoes	4000

$\mathbf{r}_2$

Emp	*Dept*	*Sal*
Ed	Shoes	1500
An	Toys	2000
Tim	Shoes	4000

$\Rightarrow$

Dept	*SUM(Sal)*
Toys	2000
Shoes	5500

Task: Return tight lower and upper bounds for *SUM(Sal)*:

Dept	*SUM(Sal)*
Toys	[2000, 3000]
Shoes	[4000, 5500]

Queries with Aggregation [ABC01]

WorksFor

Emp	*Dept*	*Sal*
Ed	Toys	1000
Ed	Shoes	1500
An	Toys	2000
Tim	Shoes	4000

SELECT Dept, SUM(Sal)
FROM WorksFor
GROUP BY Dept

$\mathbf{r}_1$

Emp	*Dept*	*Sal*
Ed	Toys	1000
An	Toys	2000
Tim	Shoes	4000

$\Rightarrow$

Dept	*SUM*(*Sal*)
Toys	3000
Shoes	4000

$\mathbf{r}_2$

Emp	*Dept*	*Sal*
Ed	Shoes	1500
An	Toys	2000
Tim	Shoes	4000

$\Rightarrow$

Dept	*SUM*(*Sal*)
Toys	2000
Shoes	5500

Task: Return tight lower and upper bounds for *SUM*(*Sal*):

Dept	*SUM*(*Sal*)
Toys	[2000, 3000]
Shoes	[4000, 5500]

Table of Contents

Denial Constraints

Definition Denial constraint

A denial constraint has the form:

$$\neg\exists\vec{x_1}\ldots\exists\vec{x_k}\,(R_1(\vec{x_1})\wedge\cdots\wedge R_k(\vec{x_k})\wedge\beta(\vec{x_1},\ldots,\vec{x_k}))$$

where β is a conjunction of atomic formulas using built-in predicates ($=$, $<$).

Denial constraints

For the schema $EMP[Name, Rank, Sal]$:

$$\neg\exists u,x,y,z\,(EMP(u,\text{`boss'},y)\wedge EMP(x,\text{`clerk'},z)\wedge y<z)$$
$$\neg\exists x,y_1,z_1,y_2,z_2\,(EMP(x,y_1,z_1)\wedge EMP(x,y_2,z_2)\wedge y_1\neq y_2)$$
$$\neg\exists x,y_1,z_1,y_2,z_2\,(EMP(x,y_1,z_1)\wedge EMP(x,y_2,z_2)\wedge z_1\neq z_2)$$

Claim: for denial constraints, $\oplus$-repairs are subset-repairs.

Conflict Hypergraph

Relative to a database **db** and a set of denial constraints.

Definition Conflict hypergraph

A conflict hypergraph is a hypergraph whose hyperedges are subsets of **db**. For every denial constraint

$$\neg\exists\vec{x_1}\ldots\exists\vec{x_k}\,(R_1(\vec{x_1})\wedge\cdots\wedge R_k(\vec{x_k})\wedge\beta(\vec{x}_1,\ldots,\vec{x}_k)) \quad ,$$

if θ is a valuation such that $\theta(\vec{x_i})=\vec{a_i}$ for $1\leq i\leq k$ and

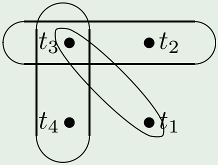
$$\mathbf{db}\models R_1(\vec{a_1})\wedge\cdots\wedge R_k(\vec{a_k})\wedge\beta(\vec{a}_1,\ldots,\vec{a}_k) \quad ,$$

then $\{R_1(\vec{a_1}),\ldots,R_k(\vec{a_k})\}$ is an hyperedge.

Example Conflict Hypergraph [CM05]

Conflict hypergraph

EMP	*Name*	*Rank*	*Sal*
t_1	Ed	clerk	28
t_2	Tim	clerk	30
t_3	An	boss	20
t_4	An	clerk	40



Properties

- Every repair is a maximal (w.r.t. $\subseteq$) subset of **db** that includes no hyperedge of the conflict hypergraph.
- The number of hyperedges is polynomial in the size of **db**.

Quantifier-Free Boolean Queries

Definition Quantifier-free Boolean query

A quantifier-free Boolean query is a Boolean combination of ground atoms. It can be assumed to be in CNF:

$$\phi_1 \land \phi_2 \land \cdots \land \phi_\ell \ ,$$

where each ϕ_i is of the form $\neg A_1 \lor \cdots \lor \neg A_m \lor B_1 \lor \cdots \lor B_n$, with $A_1, \ldots, A_m, B_1, \ldots, B_n$ distinct ground atoms.

Quantifier-free Boolean query

$\neg EMP(\text{'An'}, \text{'clerk'}, \text{'40'}) \lor EMP(\text{'Ed'}, \text{'clerk'}, \text{'28'})$

Denial Constraints and Quantifier-Free Queries

Question

The problem is to verify for $1 \leq i \leq \ell$ whether

$$\phi_i = \neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n$$

is true in every repair. We ask instead whether ϕ_i is false in some repair, i.e., whether some repair **r** satisfies

$$\neg \phi_i = A_1 \wedge \cdots \wedge A_m \wedge \neg B_1 \wedge \cdots \wedge \neg B_n \ .$$

Denial Constraints and Quantifier-Free Queries

Crux HProver algorithm [CM05, CMS04]

Any repair $\mathbf{r}$ satisfying $A_1 \wedge \cdots \wedge A_m \wedge \neg B_1 \wedge \cdots \wedge \neg B_n$ must verify the following conditions:

1. $A_1, \ldots, A_m \in \mathbf{r}$;
2. for each edge E in the conflict hypergraph, $E \not\subseteq \mathbf{r}$; and
3. Maximality: for $1 \leq j \leq n$, if $B_j \in \mathbf{db}$, then there is an edge E_j in the conflict hypergraph such that $B_j \in E_j$ and $E_j \setminus \{B_j\} \subseteq \mathbf{r}$.

Why is HProver polynomial in the size of **db**?

The Maximality condition chooses n hyperedges among a polynomial number of hyperedges.

Other Topics

- Database repairing and data exchange
- Database repairing and approximations
- Database repairing and preferences [SCM12, FKK15, KLP17, LK17]
- Database repairing and implementations
- Database repairing and database management systems
- Consistent query answering for queries with negation
- Consistent query answering in description logics
- Consistent query answering over graph databases
- ...

References I

Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki.
Scalar aggregation in fd-inconsistent databases.
In ICDT, volume 1973 of Lecture Notes in Computer Science, pages 39–53. Springer, 2001.

Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki.
Answer sets for consistent query answering in inconsistent databases.
TPLP, 3(4-5):393–424, 2003.

Serge Abiteboul, Richard Hull, and Victor Vianu.
Foundations of Databases.
Addison-Wesley, 1995.

Foto N. Afrati and Phokion G. Kolaitis.
Repair checking in inconsistent databases: algorithms and complexity.
In ICDT, volume 361 of ACM International Conference Proceeding Series, pages 31–41. ACM, 2009.

Leopoldo E. Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko.
The complexity and approximation of fixing numerical attributes in databases under integrity constraints.
Inf. Syst., 33(4-5):407–434, 2008.

Marco Calautti, Ester Livshits, Andreas Pieris, and Markus Schneider.
Counting database repairs entailing a query: The case of functional dependencies.
In PODS, pages 403–412. ACM, 2022.

Andrea Calì, Domenico Lembo, and Riccardo Rosati.
On the decidability and complexity of query answering over inconsistent and incomplete databases.
In PODS, pages 260–271. ACM, 2003.

References II

Jan Chomicki and Jerzy Marcinkowski.
Minimal-change integrity maintenance using tuple deletions.
Inf. Comput., 197(1-2):90–121, 2005.

Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko.
Computing consistent query answers using conflict hypergraphs.
In CIKM, pages 417–426. ACM, 2004.

Nilesh N. Dalvi, Christopher Ré, and Dan Suciu.
Probabilistic databases: diamonds in the dirt.
Commun. ACM, 52(7):86–94, 2009.

Sergio Flesca, Filippo Furfaro, and Francesco Parisi.
Querying and repairing inconsistent numerical databases.
ACM Trans. Database Syst., 35(2):14:1–14:50, 2010.

Ronald Fagin, Benny Kimelfeld, and Phokion G. Kolaitis.
Dichotomies in the complexity of preferred repairs.
In PODS, pages 3–15. ACM, 2015.

M. R. Garey and David S. Johnson.
Computers and Intractability: A Guide to the Theory of NP-Completeness.
W. H. Freeman, 1979.

Miika Hannula and Jef Wijsen.
A dichotomy in consistent query answering for primary keys and unary foreign keys.
In PODS, pages 437–449. ACM, 2022.

References III

Tomasz Imielinski and Witold Lipski Jr.
Incomplete information in relational databases.
J. ACM, 31(4):761–791, 1984.

Gabriele Kern-Isberner and Thomas Lukasiewicz.
Many facets of reasoning under uncertainty, inconsistency, vagueness, and preferences: A brief survey.
KI, 31(1):9–13, 2017.

Benny Kimelfeld, Ester Livshits, and Liat Peterfreund.
Detecting ambiguity in prioritized database repairing.
In ICDT, volume 68 of LIPIcs, pages 17:1–17:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

Paraschos Koutris and Jef Wijsen.
Consistent query answering for primary keys in datalog.
Theory Comput. Syst., 65(1):122–178, 2021.

Ester Livshits and Benny Kimelfeld.
Counting and enumerating (preferred) database repairs.
In PODS, pages 289–301. ACM, 2017.

Dany Maslowski and Jef Wijsen.
A dichotomy in the complexity of counting database repairs.
J. Comput. Syst. Sci., 79(6):958–983, 2013.

Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski.
Prioritized repairing and consistent query answering in relational databases.
Ann. Math. Artif. Intell., 64(2-3):209–246, 2012.

References IV

Jeffrey D. Ullman.
Principles of Database and Knowledge-Base Systems, Volume I, volume 14 of Principles of computer science series.
Computer Science Press, 1988.

Jef Wijsen.
Database repairing using updates.
ACM Trans. Database Syst., 30(3):722–768, 2005.