On Condensing Database Repairs Obtained by Tuple Deletions

Jef Wijsen

Université de Mons-Hainaut
Motivation

Several sources for database inconsistency: data integration, unenforced integrity constraints, ...

<table>
<thead>
<tr>
<th>$S$</th>
<th>Name</th>
<th>Birth</th>
<th>Sport</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bill Russell</td>
<td>1934</td>
<td>Basketball</td>
</tr>
<tr>
<td></td>
<td>Bill Russell</td>
<td>1936</td>
<td>Basketball</td>
</tr>
<tr>
<td></td>
<td>J. Anquetil</td>
<td>1934</td>
<td>Cyclist</td>
</tr>
<tr>
<td></td>
<td>R. Poulidor</td>
<td>1936</td>
<td>Cyclist</td>
</tr>
</tbody>
</table>

falsifies \( \text{Name} \rightarrow \text{Birth} \).
Motivation

Several sources for database inconsistency: data integration, unenforced integrity constraints, ...

<table>
<thead>
<tr>
<th>Name</th>
<th>Birth</th>
<th>Sport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill Russell</td>
<td>1934</td>
<td>Basketball</td>
</tr>
<tr>
<td>Bill Russell</td>
<td>1936</td>
<td>Basketball</td>
</tr>
<tr>
<td>J. Anquetil</td>
<td>1934</td>
<td>Cyclist</td>
</tr>
<tr>
<td>R. Poulidor</td>
<td>1936</td>
<td>Cyclist</td>
</tr>
</tbody>
</table>

falsifies Name $\rightarrow$ Birth.

Repairing by tuple deletions finds two possible repairs:

<table>
<thead>
<tr>
<th>Name</th>
<th>Birth</th>
<th>Sport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill Russell</td>
<td>1934</td>
<td>Basketball</td>
</tr>
<tr>
<td>J. Anquetil</td>
<td>1934</td>
<td>Cyclist</td>
</tr>
<tr>
<td>R. Poulidor</td>
<td>1936</td>
<td>Cyclist</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Birth</th>
<th>Sport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill Russell</td>
<td>1936</td>
<td>Basketball</td>
</tr>
<tr>
<td>J. Anquetil</td>
<td>1934</td>
<td>Cyclist</td>
</tr>
<tr>
<td>R. Poulidor</td>
<td>1936</td>
<td>Cyclist</td>
</tr>
</tbody>
</table>
Define a database (db) as a finite set of (not necessarily ground) atoms.
Repairs

Define a database (db) as a finite set of (not necessarily ground) atoms.

Let $\Sigma$ be a set of integrity constraints that is satisfied by the empty db.

A repair of ground db $D$ and $\Sigma$ is a db $R$ such that:

1. $R \subseteq D$ and $R \models \Sigma$; and

2. $\forall R': R \subsetneq R' \subseteq D$ implies $R' \not\models \Sigma$ (Maximality).
Consistent Query Answer

Let $\Sigma$ be a set of constraints and $D$ a (possibly inconsistent) ground db. Let $q$ be a query.

The consistent query answer to $q$ on input $D$ and $\Sigma$ is defined by:

$$q_\Sigma(D) := \bigcap \{q(R) \mid R \text{ is a repair of } D \text{ and } \Sigma \}$$

Only the tuples that are in the query answer on each repair, are certainly true.
### CQA Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Birth</th>
<th>Sport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill Russell</td>
<td>1934</td>
<td>Basketball</td>
</tr>
<tr>
<td>J. Anquetil</td>
<td>1934</td>
<td>Cyclist</td>
</tr>
<tr>
<td>R. Poulidor</td>
<td>1936</td>
<td>Cyclist</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Birth</th>
<th>Sport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill Russell</td>
<td>1936</td>
<td>Basketball</td>
</tr>
<tr>
<td>J. Anquetil</td>
<td>1934</td>
<td>Cyclist</td>
</tr>
<tr>
<td>R. Poulidor</td>
<td>1936</td>
<td>Cyclist</td>
</tr>
</tbody>
</table>

\[
\text{SameYear}(x) \leftarrow S(x, y, \text{Basketball}), S(w, y, \text{Cyclist})
\]

\[\Rightarrow \{ \text{SameYear}(\text{Bill Russell}) \} \]

\[
\text{WhichYear}(x, y) \leftarrow S(x, y, \text{Basketball}), S(w, y, \text{Cyclist})
\]

\[\Rightarrow \{ \} \]
Data Complexity

- Assume fixed schema, set $\Sigma$ of constraints, query $q$.
- Consistent query answering is the complexity of the set:

$$CQA(\Sigma, q) := \{ D \mid D \text{ is a db and } q_\Sigma(D) = \emptyset \}$$

- For a fixed set $\Sigma$ of constraints and fixed query $q$:
  given a database $D$, decide emptiness of $q_\Sigma(D)$. 
A conjunctive nucleus for $D$ and $\Sigma$ is a (not necessarily ground) database $D'$ such that for every conjunctive query $q$,

$$\text{ground}(q(D')) = q_\Sigma(D)$$

A “view” that returns consistent answers to any conjunctive query (up to removal of nonground tuples).
Conjunctive Nucleus

A **conjunctive nucleus** for $D$ and $\Sigma$ is a (not necessarily ground) database $D'$ such that for every conjunctive query $q$,

$$\text{ground}(q(D')) = q_\Sigma(D)$$

A “view” that returns consistent answers to any conjunctive query (up to removal of nonground tuples).

Compare to **query rewriting**: Given $\Sigma$ and $q$, find $q'$ such that for every database $D$,

$$q'(D) = q_\Sigma(D)$$
Conjunctive Nucleus

A conjunctive nucleus for $\mathcal{D}$ and $\Sigma$ is a (not necessarily ground) database $\mathcal{D}'$ such that for every conjunctive query $q$,

$$\text{ground}(q(\mathcal{D}')) = q_\Sigma(\mathcal{D})$$

A “view” that returns consistent answers to any conjunctive query (up to removal of nonground tuples).

Compare to query rewriting: Given $\Sigma$ and $q$, find $q'$ such that for every database $\mathcal{D}$,

$$q'(\mathcal{D}) = q_\Sigma(\mathcal{D})$$

If $\text{CQA}(\Sigma, q)$ is $\text{NP}$-hard for some conjunctive query $q$, then, assuming $\text{P} \neq \text{NP}$,

- $\mathcal{D}'$ cannot be constructed in polynomial time;
- $q'$ cannot be first-order.
### Conjunctive Nucleus Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Birth</th>
<th>Sport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill Russell</td>
<td>1934</td>
<td>Basketball</td>
</tr>
<tr>
<td>J. Anquetil</td>
<td>1934</td>
<td>Cyclist</td>
</tr>
<tr>
<td>R. Poulidor</td>
<td>1936</td>
<td>Cyclist</td>
</tr>
</tbody>
</table>

- **C**

- **SameYear**($x$) $\leftarrow S(x, y, \text{Basketball}), S(w, y, \text{Cyclist})$

  $\leadsto \{ \text{SameYear(Bill Russel)} \}$

- **WhichYear**($x, y$) $\leftarrow S(x, y, \text{Basketball}), S(w, y, \text{Cyclist})$

  $\leadsto \text{ground}\{ \text{WhichYear(Bill Russel, y)} \} = \{ \}$
Homomorphism

A homomorphism from db $C$ to db $D$ is a substitution $\theta$ such that $\theta(C) \subseteq D$.

$D \succeq C$ denotes “$C$ homomorphic to $D$.”

For example, $\{P(a, b)\} \succeq \{P(a, y), P(x, b)\}$.

$D_1 \sim D_2$ iff $D_1 \succeq D_2$ and $D_2 \succeq D_1$. 
Homomorphism

A homomorphism from db $C$ to db $D$ is a substitution $\theta$ such that $\theta(C) \subseteq D$.

$$D \geq C$$
denotes “$C$ homomorphic to $D$.”

For example, $\{P(a, b)\} \succeq \{P(a, y), P(x, b)\}$.

$D_1 \sim D_2$ iff $D_1 \succeq D_2$ and $D_2 \succeq D_1$.

Let $R$ be a finite set of databases.
An infimum of $R$ is a maximal (w.r.t. $\succeq$) db that is homomorphic to each db of $R$.

Infimum can be effectively constructed.
Assume $q$ conjunctive query, $R$ finite set of databases (repairs), and $q(R) := \{ q(R) \mid R \in R \}$.

\[
\begin{array}{c}
\inf \quad \inf(R) \\
\downarrow \quad \downarrow \\
q(R) \quad \inf(q(R)) \sim q(\inf(R))
\end{array}
\]

It follows: $\text{ground}(\inf(q(R))) = \text{ground}(q(\inf(R)))$. 
Assume $q$ conjunctive query, $\mathcal{R}$ finite set of databases (repairs), and $q(\mathcal{R}) := \{q(R) \mid R \in \mathcal{R}\}$.

It follows: $\text{ground}(\inf(q(\mathcal{R}))) = \text{ground}(q(\inf(\mathcal{R})))$.

If each db in $\mathcal{R}$ is ground, $\bigcap q(\mathcal{R}) = \text{ground}(\inf(q(\mathcal{R})))$. 

LAAIC, 26 August 2005, Copenhagen, Denmark – p.10/20
Assume $q$ conjunctive query, $\mathcal{R}$ finite set of databases (repairs), and $q(\mathcal{R}) := \{q(R) \mid R \in \mathcal{R}\}$. 

\[
\begin{array}{c}
\mathcal{R} \xrightarrow{\inf} \inf(\mathcal{R}) \\
q \downarrow \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad q \\
q(\mathcal{R}) \xrightarrow{\inf} \inf(q(\mathcal{R})) \sim q(\inf(\mathcal{R}))
\end{array}
\]

It follows: $\text{ground}(\inf(q(\mathcal{R}))) = \text{ground}(q(\inf(\mathcal{R})))$.

If each db in $\mathcal{R}$ is ground, $\bigcap q(\mathcal{R}) = \text{ground}(\inf(q(\mathcal{R})))$.

Hence, if each db in $\mathcal{R}$ is ground,

$$\bigcap q(\mathcal{R}) = \text{ground}(q(\inf(\mathcal{R})))$$
So for \( q \) conjunctive query and \( \mathcal{R} \) finite set of ground databases (repairs),

\[
\bigcap q(\mathcal{R}) = \text{ground}(q(\text{inf}(\mathcal{R})))
\]

Assume set \( \Sigma \) of constraints and db \( \mathcal{D} \). If \( \mathcal{R} \) is the set of all repairs of \( \mathcal{D} \) and \( \Sigma \), then

\[
q_\Sigma(\mathcal{D}) := \bigcap q(\mathcal{R})
\]

hence \( \text{inf}(\mathcal{R}) \) is a conjunctive nucleus for \( \mathcal{D} \) and \( \Sigma \).
Scenarios

S1. The nucleus size is not polynomially bounded in the size of the database to be repaired.

S2. The nucleus size is polynomially bounded but constructing the nucleus is not in polynomial time.

S3. Constructing a nucleus is in polynomial time.
Conjunctive Nucleus Size

Nucleus size is not polynomially bounded.
Conjunctive Nucleus Size

- Nucleus size is not polynomially bounded.
- \( \forall x \forall y \forall z (R(x, y) \land R(x, z) \rightarrow y = z) \)

![Diagram of a graph with vertices labeled 1, 2, 3, 4, n-2, n-1, n, and edges connecting them.]

- Repairs preserve one outgoing edge per vertex.
Conjunctive Nucleus Size

- Nucleus size is not polynomially bounded.

\[ \forall x \forall y \forall z (R(x, y) \land R(x, z) \rightarrow y = z) \]

- Repairs preserve one outgoing edge per vertex.
- Each repair has a cycle of size between 2 and \( n \).
- Nucleus must have a cycle of size at least \( \text{lcm}(2, 3, \ldots, n) \), which is \( \geq 2^n \) if \( n \geq 7 \).
Restricted Query Classes

- **Rule-based conjunctive query:**

  \[
  \text{Answer}(\bar{y}) \leftarrow R_1(\bar{x}_1), R_2(\bar{x}_2), \ldots, R_n(\bar{x}_n)
  \]

  where every variable in \(\bar{y}\) occurs in some \(\bar{x}_i\).

- **Linear rule:** Every variable that occurs in some \(\bar{x}_i, \bar{x}_j\) with \(i \neq j\), also occurs in \(\bar{y}\).

  \[
  \text{NonLinear}(x) \leftarrow S(x, y, \text{Basketball}), S(w, y, \text{Cyclist})
  \]

  \[
  \text{Linear}(x, y) \leftarrow S(x, y, \text{Basketball}), S(w, y, \text{Cyclist})
  \]
Restricted Query Classes

- Rule-based conjunctive query:

\[ \text{Answer}(\bar{y}) \leftarrow R_1(\bar{x}_1), R_2(\bar{x}_2), \ldots, R_n(\bar{x}_n) \]

where every variable in \( \bar{y} \) occurs in some \( \bar{x}_i \).

- Linear rule: Every variable that occurs in some \( \bar{x}_i, \bar{x}_j \) with \( i \neq j \), also occurs in \( \bar{y} \).

\[
\begin{align*}
\text{NonLinear}(x) & \leftarrow S(x, y, \text{Basketball}), S(w, y, \text{Cyclist}) \\
\text{Linear}(x, y) & \leftarrow S(x, y, \text{Basketball}), S(w, y, \text{Cyclist})
\end{align*}
\]

- Quantifier-free rule: Every variable that occurs in some \( \bar{x}_i \), also occurs in \( \bar{y} \).
Primary Keys

1. Assume $\Sigma$ expresses one primary key constraint per relation.
2. There exists a conjunctive query $q$ such that $CQA(\Sigma, q)$ is $\text{NP}$-hard.
3. Solvable in polynomial time for linear conjunctive queries.
Denials

Denials of the form:

$$\forall^* (\neg P_1(\vec{x}_1) \lor \neg P_2(\vec{x}_2) \lor \cdots \lor \neg P_n(\vec{x}_n))$$

For example:

$$\forall x \forall y_1 \forall y_2 (\neg S(x, y_1, \text{Soccer}) \lor \neg S(x, y_2, \text{Cyclist}))$$
Denials

Denials of the form:

\[ \forall^* (\neg P_1(x_1) \lor \neg P_2(x_2) \lor \cdots \lor \neg P_n(x_n)) \]

For example:

\[ \forall x \forall y_1 \forall y_2 (\neg S(x, y_1, \text{Soccer}) \lor \neg S(x, y_2, \text{Cyclist})) \]

There exists a set \( \Sigma \) of denials and a conjunctive query \( q \) such that \( \text{CQA}(\Sigma, q) \) is \textbf{NP}-hard.

Remains \textbf{NP}-hard for linear conjunctive queries!

Solvable in polynomial time for quantifier-free conjunctive queries.
Nuclei for Restricted Query Classes

Can impose further restrictions on the queries that can be answered by a nucleus.

A linear conjunctive nucleus for db $D$ and set $\Sigma$ of constraints is a (not necessarily ground) database $D'$ such that for every linear conjunctive query $q$,

$$\text{ground}(q(D')) = q_\Sigma(D)$$
Nuclei for Restricted Query Classes

- Can impose further restrictions on the queries that can be answered by a nucleus.

- A linear conjunctive nucleus for db $\mathcal{D}$ and set $\Sigma$ of constraints is a (not necessarily ground) database $\mathcal{D}'$ such that for every linear conjunctive query $q$,

$$\text{ground}(q(\mathcal{D}')) = q_{\Sigma}(\mathcal{D})$$

- Every database $\mathcal{D}$ and set $\Sigma$ of constraints (that is satisfied by the empty database) allows a linear conjunctive nucleus of polynomial size in $|\mathcal{D}|$. 
Nuclei for Restricted Query Classes

Can impose further restrictions on the queries that can be answered by a nucleus.

A linear conjunctive nucleus for db $\mathcal{D}$ and set $\Sigma$ of constraints is a (not necessarily ground) database $\mathcal{D}'$ such that for every linear conjunctive query $q$,

$$\text{ground}(q(\mathcal{D}')) = q_\Sigma(\mathcal{D})$$

Every database $\mathcal{D}$ and set $\Sigma$ of constraints (that is satisfied by the empty database) allows a linear conjunctive nucleus of polynomial size in $|\mathcal{D}|$.

For primary keys, this linear conjunctive nucleus is constructible in polynomial time.

Not so for denials $\leadsto$ quantifier-free conjunctive nucleus constructible in polynomial time.
Conclusions

- Some new [in]tractable cases of $\text{CQA}(\Sigma, q)$ identified.
- Tractable cases identified allow a nucleus constructible in polynomial time.
- Similar results obtained for “update-based” repairing.
  
  See:

## Drawback of Repairing by Deletion

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>ZIP</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ed</td>
<td>...</td>
<td>700</td>
<td>Mons</td>
</tr>
<tr>
<td>Jo</td>
<td>...</td>
<td>700</td>
<td>Mons</td>
</tr>
<tr>
<td>An</td>
<td>...</td>
<td>700</td>
<td>Bergen</td>
</tr>
</tbody>
</table>

falsifies ZIP → City.

Repairing by deletions will find two possible repairs:

\[ R_1 \]
<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>ZIP</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ed</td>
<td>...</td>
<td>700</td>
<td>Mons</td>
</tr>
<tr>
<td>Jo</td>
<td>...</td>
<td>700</td>
<td>Mons</td>
</tr>
</tbody>
</table>

\[ R_2 \]
<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>ZIP</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>An</td>
<td>...</td>
<td>700</td>
<td>Bergen</td>
</tr>
</tbody>
</table>

Much information lost (imagine “long” tuples).
Repairing by Updates

<table>
<thead>
<tr>
<th>$P$</th>
<th>N</th>
<th>ZIP</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ed</td>
<td>...</td>
<td>700</td>
<td>Mons</td>
</tr>
<tr>
<td>Jo</td>
<td>...</td>
<td>700</td>
<td>Mons</td>
</tr>
<tr>
<td>An</td>
<td>...</td>
<td>700</td>
<td>Bergen</td>
</tr>
</tbody>
</table>

falsifies ZIP → City.

Instead of e.g. deleting the third tuple (see $R_1$ above), conclude that Bergen or 700 are mistaken:

<table>
<thead>
<tr>
<th>$R_1'$</th>
<th>N</th>
<th>ZIP</th>
<th>City</th>
<th>$R_1''(x)$</th>
<th>N</th>
<th>ZIP</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ed</td>
<td>...</td>
<td>700</td>
<td>Mons</td>
<td>$x \neq 700$</td>
<td>Ed</td>
<td>...</td>
<td>700</td>
</tr>
<tr>
<td>Jo</td>
<td>...</td>
<td>700</td>
<td>Mons</td>
<td></td>
<td>Jo</td>
<td>...</td>
<td>700</td>
</tr>
<tr>
<td>An</td>
<td>...</td>
<td>700</td>
<td>Mons</td>
<td></td>
<td>An</td>
<td>...</td>
<td>$x$</td>
</tr>
</tbody>
</table>

Keep error-free components of tuples.