# CS294-248 Special Topics in Database Theory Unit 6: Constraints, Incomplete and Probabilistic Databases (Part 2)

Dan Suciu

University of Washington

# Outline

- Tuesday: Generalized Constraints, Semantics Optimization.

- Today: Repairs, Incomplete Databases

# Recap: Generalized Dependencies

Tuple-Generating Dependency (TGD):

$$\boxed{\forall \boldsymbol{x}\,(A_1 \wedge \ldots \wedge A_m \Rightarrow \exists \boldsymbol{y}(B_1 \wedge \cdots \wedge B_k))}$$

The TGD is full if there is no $\exists \boldsymbol{y}$

Equality-Generating Dependency (EGD):

$$\boxed{\forall \boldsymbol{x}(A_1 \wedge \ldots \wedge A_m \Rightarrow x_i = x_j)}$$

# Recap: Chase

$Q \stackrel{\sigma,\theta}{\to} Q'$, where

- If $\sigma \equiv \forall \boldsymbol{x}(A \Rightarrow \exists \boldsymbol{y} B)$, then $Q' = Q \wedge \theta(B)$.
- If $\sigma \equiv \forall \boldsymbol{x}(A \Rightarrow (x_i = x_j))$, then $Q' = Q[x_j/x_i]$.

Key property: $\sigma \models Q \equiv Q'$.

# Repairs for FDs

# Definition

Consider a set of constraints $\Sigma$ and a database $\boldsymbol{D}$.

$\boldsymbol{D} \not\models \Sigma$.

**The Database Repair Problem**

Find another database $\boldsymbol{D}'$ such that $\boldsymbol{D}' \models \Sigma$ and $|\boldsymbol{D} \Delta \boldsymbol{D'}|$ is minimal.

(Recall: $S_1 \Delta S_2 = (S_1 - S_2) \cup (S_2 - S_1)$.)

Equivalently: perform a minimum number of updates to satisfy $\Sigma$.

# The FD-Repair Problem

$\Sigma$ is a set of FDs

The updates are restricted to be be deletions

Given $\boldsymbol{D}$, delete minimum number of tuples to obtain $\boldsymbol{D}' \subseteq \boldsymbol{D}$ and $\boldsymbol{D}' \models \Sigma$.

We study the complexity as a function of $|\boldsymbol{D}|$.

# Example 1: Repairing $A \to B$

$\boxed{A \to B}$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

# Example 1: Repairing $A \to B$

$\boxed{A \to B}$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Group the tuples by $A$

In each group $a_1, a_2, \ldots$ keep only one $b_j$ (the most frequent).

# Example 1: Repairing $A \to BC$

$\boxed{A \to BC}$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

# Example 1: Repairing $A \to BC$

$\boxed{A \to BC}$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

Same as before: treat $BC$ as a single attribute.

# Example 3: $A \to B \to C$

$\boxed{A \to B \to C}$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

# Example 3: $A \to B \to C$

$\boxed{A \to B \to C}$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

This is NP-hard!

Reduction from Max-SAT

**Theorem ([Williams, 2016])**

*The problem given a 2CNF, check $\geq 7/10$ clauses can be satisfied is NP-complete.*

# Proof for $A \to B \to C$

Start with a 2CNF formula $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$

Create a relation instance $R(A, B, C)$ as follows:

# Proof for $A \to B \to C$

Start with a 2CNF formula $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$

Create a relation instance $R(A, B, C)$ as follows:

For each clause $C_i = ((\neg)X \vee (\neg)Y)$ add two tuples to $R$

- Tuple $(i, X, 0)$ or $(i, X, 1)$, depending on whether $\neg X$ or $X$
- Tuple $(i, Y, 0)$ or $(i, Y, 1)$, depending on whether $\neg Y$ or $Y$

# Proof for $A \to B \to C$

Start with a 2CNF formula $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$

Create a relation instance $R(A, B, C)$ as follows:

For each clause $C_i = ((\neg)X \vee (\neg)Y)$ add two tuples to $R$

- Tuple $(i, X, 0)$ or $(i, X, 1)$, depending on whether $\neg X$ or $X$
- Tuple $(i, Y, 0)$ or $(i, Y, 1)$, depending on whether $\neg Y$ or $Y$

**Claim** $\geq 7n/10$ clauses can be satisfied iff $\exists$ repair of size $\geq 7n/10$.

# Proof for $A \to B \to C$

Start with a 2CNF formula $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$

Create a relation instance $R(A, B, C)$ as follows:

For each clause $C_i = ((\neg)X \vee (\neg)Y)$ add two tuples to $R$

- Tuple $(i, X, 0)$ or $(i, X, 1)$, depending on whether $\neg X$ or $X$
- Tuple $(i, Y, 0)$ or $(i, Y, 1)$, depending on whether $\neg Y$ or $Y$

**Claim** $\geq 7n/10$ clauses can be satisfied iff $\exists$ repair of size $\geq 7n/10$.

**Proof** $A \to B$ ensures that we retain $\leq 1$ tuple per clause

$B \to C$ ensures that we assign consistent values to the same variable.

# Discussion so Far

$A \to B$ in PTIME

$A \to BC$ in PTIME

$A \to B \to C$ NP-hard

What's the general rule?

# Unusual FDs

We are familiar with $AB \rightarrow CD$ or $A \rightarrow C$.

What does $A \rightarrow \emptyset$ mean?

# Unusual FDs

We are familiar with $AB \to CD$ or $A \to C$.

What does $A \to \emptyset$ mean?

It is always true.

# Unusual FDs

We are familiar with $AB \to CD$ or $A \to C$.

What does $A \to \emptyset$ mean? It is always true.

What does $\emptyset \to A$ mean?

# Unusual FDs

We are familiar with $AB \to CD$ or $A \to C$.

What does $A \to \emptyset$ mean? It is always true.

What does $\emptyset \to A$ mean? $A$ has a single value.

# Example 4: $\emptyset \to A$

$\boxed{\emptyset \to A}$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

# Example 4: $\emptyset \to A$

$\boxed{\emptyset \to A}$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

We keep a single value of $A$, namely the most frequent one.

# Example 4: $\emptyset \to A$

$\boxed{\emptyset \to A}$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

We keep a single value of $A$, namely the most frequent one.

Now consider:

$\emptyset \to A$
$B \to C$

Compute optimal repair. How?

# Example 4: $\emptyset \to A$

$\boxed{\emptyset \to A}$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

We keep a single value of $A$, namely the most frequent one.

Now consider:

$\emptyset \to A$
$B \to C$

Compute optimal repair. How?

For each $A = a_i$ compute optimal repair of $B \to C$, keep the largest.

# Example 4: $\emptyset \to A$

$\boxed{\emptyset \to A}$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

We keep a single value of $A$, namely the most frequent one.

Now consider:

$\emptyset \to A$
$B \to C$

Compute optimal repair. How?

For each $A = a_i$ compute optimal repair of $B \to C$, keep the largest.

Consensus rule: if $\Sigma$ contains $\emptyset \to A$, then compute the optimal repair for each value $A = a_1, a_2 \ldots$, return the largest.

# Example 5

$$\boxed{\begin{array}{l} A \rightarrow B \\ AC \rightarrow D \end{array}}$$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

# Example 5

$$\boxed{\begin{array}{l} A \rightarrow B \\ AC \rightarrow D \end{array}}$$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

For each value $A = a_i$, compute the optimal repair of the residual:

$$\boxed{\begin{array}{l} \emptyset \rightarrow B \\ C \rightarrow D \end{array}}$$

Use the consensus rule.

# Example 5

$$\boxed{\begin{array}{l} A \rightarrow B \\ AC \rightarrow D \end{array}}$$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

For each value $A = a_i$, compute the optimal repair of the residual:

$$\boxed{\begin{array}{l} \emptyset \rightarrow B \\ C \rightarrow D \end{array}}$$

Use the consensus rule.

Compute optimal repair. How?

Common LHS rule: if all LHS contain $A$, $\Sigma = \{AX_1 \rightarrow Y_1, AX_2 \rightarrow Y_2, \ldots\}$, then repair separately each $A = a_i$.

# Example 6

$$A \to B$$
$$B \to A$$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

# Example 6

$$\boxed{\begin{array}{l} A \rightarrow B \\ B \rightarrow A \end{array}}$$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

Find a maximal matching the bipartite graph $(A, B, \Pi_{AB}(R))$.

A maximal matching in a bipartite graph can be found in PTIME using the "Hungarian Algorithm".

# Last Example

$$A \to B$$
$$B \to A$$
$$AB \to C$$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

# Last Example

$$\boxed{\begin{array}{l} A \rightarrow B \\ B \rightarrow A \\ AB \rightarrow C \end{array}}$$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

For each pair $A = a_i, B = b_j$ compute optimal repair.

Weight of edge $(a_i, b_j)$ is the size of the repair.

Find a maximal weighted matching in bipartite graph.

# Last Example

$A \to B$
$B \to A$
$AB \to C$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_1$ | $\cdots$ |
| $a_1$ | $b_2$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_1$ | $\cdots$ |
| $a_2$ | $b_1$ | $c_2$ | $\cdots$ |
| $a_2$ | $b_2$ | $c_3$ | $\cdots$ |
| $a_3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| | $\cdots$ | | |

Compute optimal repair. How?

Marriage Rule

For each pair $A = a_i, B = b_j$ compute optimal repair.

Weight of edge $(a_i, b_j)$ is the size of the repair.

Find a maximal weighted matching in bipartite graph.

# The Algorithm

[Livshits et al., 2020]
Given $\Sigma, R$, compute minimal repair that satisfies $\Sigma$.

- If $\Sigma = \emptyset$ then return $R$.
- Common LHS Rule If all LHS contain $A$, then repair each $A = a_i$. Return their union.

# The Algorithm

[Livshits et al., 2020]
Given $\Sigma, R$, compute minimal repair that satisfies $\Sigma$.

- If $\Sigma = \emptyset$ then return $R$.
- Common LHS Rule If all LHS contain $A$, then repair each $A = a_i$.
  Return their union.
- Consensus Rule If $\Sigma$ contains $\emptyset \to A$, then repair each $A = a_i$.
  Return the best repair.

# The Algorithm

[Livshits et al., 2020]
Given $\Sigma$, $R$, compute minimal repair that satisfies $\Sigma$.

- If $\Sigma = \emptyset$ then return $R$.
- Common LHS Rule If all LHS contain $A$, then repair each $A = a_i$.
  Return their union.
- Consensus Rule If $\Sigma$ contains $\emptyset \to A$, then repair each $A = a_i$.
  Return the best repair.
- Marriage Rule If $\boldsymbol{U}^+ = \boldsymbol{V}^+$ and every rule has on the LHS either $\boldsymbol{U}$ or $\boldsymbol{V}$, then compute optimal repair for all pairs $\boldsymbol{U} = \boldsymbol{u}_i$, $\boldsymbol{V} = \boldsymbol{v}_j$
  Return maximal matching in weighted bipartite graph.

# The Algorithm

[Livshits et al., 2020]
Given $\Sigma, R$, compute minimal repair that satisfies $\Sigma$.

- If $\Sigma = \emptyset$ then return $R$.
- Common LHS Rule If all LHS contain $A$, then repair each $A = a_i$.
  Return their union.
- Consensus Rule If $\Sigma$ contains $\emptyset \to A$, then repair each $A = a_i$.
  Return the best repair.
- Marriage Rule If $\boldsymbol{U}^+ = \boldsymbol{V}^+$ and every rule has on the LHS either $\boldsymbol{U}$ or $\boldsymbol{V}$, then compute optimal repair for all pairs $\boldsymbol{U} = \boldsymbol{u}_i$, $\boldsymbol{V} = \boldsymbol{v}_j$
  Return maximal matching in weighted bipartite graph.
- None of the above? Fail The problem is NP-hard.

# Discussion

- Repairing for FDs: Dichotomy Theorem in [Livshits et al., 2020]. For each $\Sigma$, the the problem is either in PTIME or NP-hard.
- Data Exchange. Constraints are TGDs, LHS restricted to an input source database, RHS restricted to a target database. The repair is done via chase.
- A few other hardness results are known for repairing specific constraints (e.g. denial constraints).
- Related to the MAP problem in graphical models.

# Incomplete Databases

# Incomplete Databases

- A simple, pure theoretical concept that allows us to reason about different possible states of the database.
- Originally introduced by Imielinski and Lipski [Imielinski and Jr., 1984].
- I used these references: [Abiteboul et al., 1995, Chap.19], [Green and Tannen, 2006], [Libkin, 2014].

# Definition

Recall: a database instance is $\boldsymbol{D} = (R_1^D, R_2^D, \ldots)$.

Let $\mathcal{N}$ be the set of all database instances.

**Definition**

An incomplete database is a set $\mathcal{I} \subseteq \mathcal{N}$.

$\mathcal{I} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots\}$ the database instance can be in one of several states.

Possible Worlds, PWD.

# Problems

How do we represent an incomplete database compactly?

How do we compute queries over incomplete databases?

# Representation

# Representations

- Codd tables.
- v-tables of naive tables.
- c-tables or conditional-tables. Special case:
  - ?-tables
  - or-tables

# Representations

- Codd tables.
- v-tables of naive tables. We start here
- c-tables or conditional-tables. Special case:
  - ?-tables
  - or-tables

# v-Tables

Dom = and infinite domain of values: $a, b, c, \ldots$

Null = an infinite set of marked NULLs: $\perp_1, \perp_2, \ldots$

# v-Tables

Dom = and infinite domain of values: $a, b, c, \ldots$
Null = an infinite set of marked NULLs: $\perp_1, \perp_2, \ldots$

## Definition

A v-table (a.k.a. naive table) is a finite set $R^I \subseteq (\text{Dom} \cup \text{Null})^k$.
Its semantics is: $[[R^I]] = \{\nu(R^I) \mid \nu : \text{Null} \rightarrow \text{Dom}\}$.

# v-Tables

Dom = and infinite domain of values: $a, b, c, \ldots$
Null = an infinite set of marked NULLs: $\perp_1, \perp_2, \ldots$

## Definition

A v-table (a.k.a. naive table) is a finite set $R^I \subseteq (\text{Dom} \cup \text{Null})^k$.
Its semantics is: $[[R^I]] = \{\nu(R^I) \mid \nu : \text{Null} \rightarrow \text{Dom}\}$.

**Example** $R^I =$

| Name | City |
|---|---|
| Alice | $\perp_1$ |
| Bob | SF |
| Carol | $\perp_2$ |
| Dave | $\perp_1$ |

What is $[[R^I]]$?

# v-Tables

Dom = and infinite domain of values: $a, b, c, \ldots$
Null = an infinite set of marked NULLs: $\perp_1, \perp_2, \ldots$

## Definition

A v-table (a.k.a. naive table) is a finite set $R^I \subseteq (\text{Dom} \cup \text{Null})^k$.
Its semantics is: $[[R^I]] = \{\nu(R^I) \mid \nu : \text{Null} \rightarrow \text{Dom}\}$.

**Example** $R^I =$

| Name | City |
|---|---|
| Alice | $\perp_1$ |
| Bob | SF |
| Carol | $\perp_2$ |
| Dave | $\perp_1$ |

What is $[[R^I]]$?

| Name | City |
|---|---|
| Alice | $a$ |
| Bob | SF |
| Carol | $a$ |
| Dave | $a$ |

| Name | City |
|---|---|
| Alice | $a$ |
| Bob | SF |
| Carol | $b$ |
| Dave | $a$ |

| Name | City |
|---|---|
| Alice | $a$ |
| Bob | SF |
| Carol | $c$ |
| Dave | $a$ |

...

Single restriction: Alice and Dave are in the same "City".

# Codd Tables

**Definition**

A Codd table is a v-table where all marked nulls are distinct.

# Codd Tables

**Definition**

A Codd table is a v-table where all marked nulls are distinct.

**Example** $R^I =$

| Name | City |
|---|---|
| Alice | $\perp_1$ |
| Bob | SF |
| Carol | $\perp_2$ |
| Dave | $\perp_3$ |

What is $[[R^I]]$?

# Codd Tables

**Definition**

A Codd table is a v-table where all marked nulls are distinct.

**Example** $R^I =$

| Name | City |
|---|---|
| Alice | $\perp_1$ |
| Bob | SF |
| Carol | $\perp_2$ |
| Dave | $\perp_3$ |

What is $[[R^I]]$?

Same as before, but now there is no restriction for Alice and Dave to be in the same city.

# C-Tables

**Definition**

A C-table is a v-table where tuples are annotated with Boolean formulas.

The Boolean formulas use a set of Boolean variables, and/or atoms of the form $\perp_i = \perp_j$ or $\perp_i = \text{const}$.

# C-Tables

## Definition

A C-table is a v-table where tuples are annotated with Boolean formulas.

The Boolean formulas use a set of Boolean variables, and/or atoms of the form $\perp_i = \perp_j$ or $\perp_i =$ const.

**Example** $R^I =$

| Name | City | |
|---|---|---|
| Alice | $\perp_1$ | $X_1$ |
| Bob | SF | $X_1 \wedge (\perp_2 = \text{'SF'})$ |
| Carol | $\perp_2$ | $\texttt{true}$ |
| Dave | $\perp_1$ | $X_2$ |

Alice, Bob present only if $X_1 = \texttt{true}$.

Bob is present only if, in addition, Carol lives in SF

Dave is present only if $X_2 = \texttt{true}$.

# Special case of C-Tables: Maybe Tables

## Definition

A maybe-table, or ?-table is a conventional table $R^I$ where each tuple is annotated by a ?. Semantics: $[[R^I]] = \{R \mid R \subseteq R^I\}$.

# Special case of C-Tables: Maybe Tables

## Definition

A maybe-table, or ?-table is a conventional table $R^I$ where each tuple is annotated by a ?. Semantics: $[[R^I]] = \{R \mid R \subseteq R^I\}$.

**Example** $R^I =$

| Name | City | |
|---|---|---|
| Alice | Seattle | ? |
| Bob | SF | ? |
| Carol | Boston | ? |
| Dave | Seattle | ? |

Semantics: $\mathcal{P}(R^I)$ (16 possible worlds).

This is a special of a c-table. Why?

# Special case of C-Tables: OR-Table

**Definition**

An or-table is like a conventional table where each value can be an or-set.

An or-set, is a set whose meaning is "exactly one of its elements".
E.g. $\langle a, b, c \rangle$ means $a$ or $b$ or $c$.

# Special case of C-Tables: OR-Table

## Definition

An or-table is like a conventional table where each value can be an or-set.

An or-set, is a set whose meaning is "exactly one of its elements".
E.g. $\langle a, b, c\rangle$ means $a$ or $b$ or $c$.

**Example** $R^I =$

What is $[[R^I]]$?

| Name | City |
|---|---|
| Alice | ⟨SF, Boston⟩ |
| Bob | SF |
| Carol | Boston |
| Dave | ⟨Seattle, SF⟩ |

# Special case of C-Tables: OR-Table

## Definition

An or-table is like a conventional table where each value can be an or-set.

An or-set, is a set whose meaning is "exactly one of its elements".
E.g. $\langle a, b, c\rangle$ means $a$ or $b$ or $c$.

**Example** $R^I =$

| Name | City |
|---|---|
| Alice | ⟨SF, Boston⟩ |
| Bob | SF |
| Carol | Boston |
| Dave | ⟨Seattle, SF⟩ |

What is $[[R^I]]$?

| Name | City |
|---|---|
| Alice | SF |
| Bob | SF |
| Carol | Boston |
| Dave | Seattle |

| Name | City |
|---|---|
| Alice | SF |
| Bob | SF |
| Carol | Boston |
| Dave | SF |

| Name | City |
|---|---|
| Alice | Boston |
| Bob | SF |
| Carol | Boston |
| Dave | Seattle |

| Name | City |
|---|---|
| Alice | Boston |
| Bob | SF |
| Carol | Boston |
| Dave | SF |

# Discussion

- Incomplete databases are a very general abstraction, meant to capture several scenarios:
  - Standard NULLs define an incomplete database.
  - Repairs for FDs can be described as an incomplete database.
  - Or-sets are a natural way to express alternatives.
- We saw incomplete *tables*; this extends to incomplete *databases*.
- We used the Closed World Assumption, CWA.
  Alternative: Open World Assumption, OWA.
- An incomplete database system: [Antova et al., 2007].

# Queries on Incomplete Databases

# Querying an Incomplete Database

Fix a query $Q$.

**Definition**

If $\mathcal{I} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots\}$ is an incomplete database, then

$$Q(\mathcal{I}) \stackrel{\text{def}}{=} \{Q(\boldsymbol{D}_1), Q(\boldsymbol{D}_2), \ldots\}$$

How do we represent $Q(\mathcal{I})$?

# Closed Representation System

Fix a representation system $\mathcal{R}$ (e.g. v-tables)
and a query language $\mathcal{L}$ (e.g. CQ or FO).

## Definition

$\mathcal{R}$ is closed under $\mathcal{L}$, if for any $\boldsymbol{D}^I \in \mathcal{R}$ and any query $Q \in \mathcal{L}$, there exists a representation $A^I$ for the query answer, in other words $[[A^I]] = Q([[\boldsymbol{D}^I]])$.

# Closed Representation Systems

## Fact

V-tables are not closed under FO:

**Proof** $Q(X) = R(X) \wedge \neg S(X)$, $R = \{1, 2\}$, $S^I = \{\perp\}$ .
Then $Q([[R, S^I]]) = \{\{1, 2\}, \{1\}, \{2\}\}$; not representable as a v-table.

# Closed Representation Systems

### Fact

V-tables are not closed under FO:

**Proof** $Q(X) = R(X) \wedge \neg S(X)$, $R = \{1,2\}$, $S^I = \{\perp\}$ .
Then $Q([[R, S^I]]) = \{\{1,2\}, \{1\}, \{2\}\}$; not representable as a v-table.

### Theorem

*C-tables are closed under FO.*

# Discussion

Computing and representing all possible answers $Q(\mathcal{I})$ is difficult, and often not very informative.

A better alternative: certain answers

Also an option (but less desirable): possible answers

# Certain Answers and Possible Answers

## Definition

A *certain tuple* is a tuple $t$ s.t. $\forall \boldsymbol{D} \in \mathcal{I}$, $t \in Q(\boldsymbol{D})$. Their set: $\texttt{cert}(Q, \mathcal{I})$
A *possible tuple* is a tuple $t$ s.t. $\exists \boldsymbol{D} \in \mathcal{I}$, $t \in Q(\boldsymbol{D})$. Their set: $\texttt{poss}(Q, \mathcal{I})$

Equivalently:

$$\texttt{cert}(Q, \mathcal{I}) = \bigcap \{Q(\boldsymbol{D}) \mid \boldsymbol{D} \in \mathcal{I}\}$$
$$\texttt{poss}(Q, \mathcal{I}) = \bigcup \{Q(\boldsymbol{D}) \mid \boldsymbol{D} \in \mathcal{I}\}$$

# Example

Querying v-tables:

$$R^I = \begin{array}{|c|c|} \hline x & \perp_1 \\ y & \perp_1 \\ z & \perp_2 \\ \hline \end{array} \qquad S^I = \begin{array}{|c|c|} \hline \perp_1 & a \\ \perp_2 & b \\ \perp_2 & c \\ \perp_3 & d \\ \hline \end{array}$$

$$Q(X, Z) = R(X, Y) \wedge S(Y, Z)$$

What are the certain tuples? The possible tuples?

# Example

Querying v-tables:

$$R^I = \begin{array}{|c|c|} \hline x & \perp_1 \\ y & \perp_1 \\ z & \perp_2 \\ \hline \end{array} \qquad S^I = \begin{array}{|c|c|} \hline \perp_1 & a \\ \perp_2 & b \\ \perp_2 & c \\ \perp_3 & d \\ \hline \end{array}$$

$Q(X, Z) = R(X, Y) \wedge S(Y, Z)$

What are the certain tuples? The possible tuples?

$\texttt{cert}(Q, \mathcal{I}) =$

| | |
|---|---|
| $x$ | $a$ |
| $y$ | $a$ |
| $z$ | $b$ |
| $z$ | $c$ |

$\texttt{poss}(Q, \mathcal{I}) =$

| | |
|---|---|
| $x$ | $a$ |
| $x$ | $b$ |
| $\ldots$ | |
| $z$ | $d$ |

The cartesian product.

# Strong/Weak Representation Systems

Following [Libkin, 2014].

Fix a representation system $\mathcal{R}$, query language $\mathcal{L}$.

$\mathcal{R}$ is a strong representation system for $\mathcal{L}$ if it is closed under $\mathcal{L}$, i.e. for all $\boldsymbol{D}^I \in \mathcal{R}, Q \in \mathcal{L}$, $\exists A^I \in \mathcal{R}$ such that:

$$[[A^I]] = \{Q(\boldsymbol{D}) \mid \boldsymbol{D} \in [[\boldsymbol{D}^I]]\}$$

$\mathcal{R}$ is a weak representation system for $\mathcal{L}$ if for all $\boldsymbol{D}^I \in \mathcal{R}, Q \in \mathcal{L}$, $\exists A^I \in \mathcal{R}$ such that, for all $q \in \mathcal{L}$

$$\texttt{cert}(q, [[A^I]]) = \texttt{cert}(q, \{Q(\boldsymbol{D}) \mid \boldsymbol{D} \in [[\boldsymbol{D}^I]]\})$$

In other words, we cannot represent the possible answers exactly, but we can represent all the certain answers on all future queries $q$.

# V-Tables are a Weak Representation System for UCQs

**Theorem**

*V-tables are a weak representation system for UCQs.*

# V-Tables are a Weak Representation System for UCQs

## Theorem

*V-tables are a weak representation system for UCQs.*

$$R^I = \begin{array}{|c|c|}\hline x & \perp_1 \\ y & \perp_1 \\ z & \perp_2 \\ \hline\end{array} \quad S^I = \begin{array}{|c|c|}\hline \perp_1 & a \\ \perp_2 & b \\ \perp_2 & c \\ \perp_3 & d \\ \hline\end{array}$$

$$Q(X, Y, Z) = R(X, Y) \wedge S(Y, Z)$$

# V-Tables are a Weak Representation System for UCQs

## Theorem

*V-tables are a weak representation system for UCQs.*

$$R^I = \begin{array}{|c|c|} x & \perp_1 \\ y & \perp_1 \\ z & \perp_2 \end{array} \quad S^I = \begin{array}{|c|c|} \perp_1 & a \\ \perp_2 & b \\ \perp_2 & c \\ \perp_3 & d \end{array}$$

$$Q(X,Y,Z) = R(X,Y) \wedge S(Y,Z)$$

$$Q(R^I, S^I) = \begin{array}{|c|c|c|} x & \perp_1 & a \\ y & \perp_1 & a \\ z & \perp_2 & b \\ z & \perp_2 & c \end{array}$$

# Discussion

- Does SQL adopt the possible world semantics of Codd tables?

# Discussion

- Does SQL adopt the possible world semantics of Codd tables?

  NO: if city $= NULL$ then city $=' SF'$ `or` city$! =' SF'$ should be `true`, but in SQL it is `unknown`.

# Discussion

- Does SQL adopt the possible world semantics of Codd tables?

  NO: if city $= NULL$ then city $=' SF'$ `or` city! $=' SF'$ should be `true`, but in SQL it is `unknown`.

- What is the complexity of computing $\texttt{cert}(Q, \boldsymbol{D}^I)$ when $Q$ is a CQ and $\boldsymbol{D}^I$ is a v-database?

# Discussion

- Does SQL adopt the possible world semantics of Codd tables?

  NO: if city $= NULL$ then city $='SF'$ `or` city! $='SF'$ should be `true`, but in SQL it is `unknown`.

- What is the complexity of computing $\texttt{cert}(Q, \boldsymbol{D}^I)$ when $Q$ is a CQ and $\boldsymbol{D}^I$ is a v-database?

  In PTIME! Compute $Q$ naively on the representation, return tuples that don't have a $\perp$.

# Discussion

- Does SQL adopt the possible world semantics of Codd tables?

  NO: if city $= NULL$ then city $='SF'$ `or` city! $='SF'$ should be `true`, but in SQL it is `unknown`.

- What is the complexity of computing $\texttt{cert}(Q, \boldsymbol{D}^I)$ when $Q$ is a CQ and $\boldsymbol{D}^I$ is a v-database?

  In PTIME! Compute $Q$ naively on the representation, return tuples that don't have a $\perp$.

- **Theorem** when $Q$ is in FO, then the complexity of $\texttt{cert}(Q, \boldsymbol{D}^I)$ where $\boldsymbol{D}^I$ is a v-database is co-NP hard.

# Announcement

- No lectures next week! Join the workshop at Simons.

- The following week: two guest lectures by Val Tannen on semirings and their applications to databases.

Abiteboul, S., Hull, R., and Vianu, V. (1995).
*Foundations of Databases*.
Addison-Wesley.

Antova, L., Koch, C., and Olteanu, D. (2007).
From complete to incomplete information and back.
In Chan, C. Y., Ooi, B. C., and Zhou, A., editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 713–724. ACM.

Green, T. J. and Tannen, V. (2006).
Models for incomplete and probabilistic information.
*IEEE Data Eng. Bull.*, 29(1):17–24.

Imielinski, T. and Jr., W. L. (1984).
Incomplete information in relational databases.
*J. ACM*, 31(4):761–791.

Libkin, L. (2014).
Incomplete data: what went wrong, and how to fix it.
In Hull, R. and Grohe, M., editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 1–13. ACM.

Livshits, E., Kimelfeld, B., and Roy, S. (2020).
Computing optimal repairs for functional dependencies.
*ACM Trans. Database Syst.*, 45(1):4:1–4:46.

Williams, R. (2016).
Exact algorithms for maximum two-satisfiability.
In *Encyclopedia of Algorithms*, pages 683–688.