Dependencies Revisited for Improving Data Quality

Wenfei Fan

University of Edinburgh &

Bell Laboratories
Real-world data is often **dirty**

Ms. Stone, according to our database records you are supposed to be dead
Real-world data is often **dirty**

Ms. Stone, according to our database records you are supposed to be dead

- US: Pentagon asked **275** dead/wounded officers to re-enlist
- UK: there are **81 million** national insurance numbers but only **60 million** people eligible
- Australia: **500,000** dead people retain active medicare cards
- In a database of **500,000** customers, **120,000** records become invalid within a year – death, divorce, marriage, move
- typical data error rate in industry: **1% – 5%**, up to **30%**
- . . .

**Errors, conflicts and inconsistencies**
Dirty data is costly

- Poor data costs US companies $600 billion annually;
- Erroneously priced data in retail databases costs US customers $2.5 billion each year;
- World-wide losses from payment card fraud reached $4.84 billion in 2006;
- 30% – 80% of the development time for data cleaning in a data integration project; and
- don’t forget “dirty data” about WMD in Iraq

The market for data quality tools is growing at 17% annually
$$\gg$$ 7% average of IT segments
Research activities

Statistics, management, and computer science

- **Error correction** (data imputation): to localize tuples that violate a given set of semantic rules, and fix erroneous values in the tuples that are identified as violations of the rules.

- **Object identification**: to identify tuples from one or more relations that refer to the same real-world object.

- **Profiling**: to infer and discover meta-data (constraints or semantic rules) from sample data.

- **Data integration**: to resolve conflicts in the sources via object identification; quality-driven query processing by explicitly taking into account the quality of data from various sources.

Approaches: probabilistic, empirical, rule-based, and logic-based, ...
A principled approach based on data dependencies

A promising approach, logic-based

- Capturing a fundamental part of the semantics of data: inconsistencies emerge as violations of dependencies
- Reasoning about the semantics of the data: inference systems, analysis algorithms, ...
- Semantic profiling: discovery of dependencies for error correction and object identification

...
A principled approach based on data dependencies

A promising approach, logic-based

- Capturing a fundamental part of the semantics of data: inconsistencies emerge as violations of dependencies
- Reasoning about the semantics of the data: inference systems, analysis algorithms, ...
- Semantic profiling: discovery of dependencies for error correction and object identification

Dependencies considered for data cleaning: often traditional
- functional dependencies,
- inclusion dependencies

... designed for improving the quality of schema
A principled approach based on data dependencies

A promising approach, logic-based

- Capturing a fundamental part of the semantics of data: inconsistencies emerge as violations of dependencies
- Reasoning about the semantics of the data: inference systems, analysis algorithms, ...
- Semantic profiling: discovery of dependencies for error correction and object identification

Dependencies considered for data cleaning: often traditional

- functional dependencies,
- inclusion dependencies

... designed for improving the quality of schema

Revising traditional dependencies, for improving data quality
Conditional dependencies for capturing data inconsistencies
  - Conditional functional dependencies (CFDs)
  - Conditional inclusion dependencies (CINDs)
  - Other extensions

Matching dependencies for object identification

Static analyses: New challenges
  - Reasoning about conditional dependencies:
    - Inferring matching rules

Improving data quality with dependencies
  - Data repairing (Arenas, Bertossi, Chomicki)
  - Consistent querying answering (Arenas, Bertossi, Chomicki)
  - Condensed representations of all repairs

Open research issues

Joint work with Philip Bohannon, Loreto Bravo, Gao Cong, Floris Geerts, Xibei Jia, Anastasios Kementsietsidis, Shuai Ma
Surveys on data quality

- ... 

Sources of the statistics

Outline

- Conditional dependencies for capturing data inconsistencies
  - Conditional functional dependencies (CFDs)
  - Conditional inclusion dependencies (CINDs)
  - Other extensions

- Matching dependencies for object identification

- Static analyses: New challenges
  - Reasoning about conditional dependencies:
  - Inferring matching rules

- Improving data quality with dependencies
  - Data repairing (Arenas, Bertossi, Chomicki)
  - Consistent querying answering (Arenas, Bertossi, Chomicki)
  - Condensed representations of all repairs

- Open research issues
Example: customer relation

One of the **central technical problems** is how to tell whether the data is dirty or clean

- **Schema:** country code (CC), area code (AC), phone (phn), ...

  $$\text{Cust}(\text{CC}: \text{int}, \text{AC}: \text{int}, \text{phn}: \text{int}, \text{name}: \text{string}, \text{street}: \text{string}, \text{city}: \text{string}, \text{zip}: \text{string})$$

- **Instance:**

<table>
<thead>
<tr>
<th></th>
<th>CC</th>
<th>AC</th>
<th>phn</th>
<th>name</th>
<th>street</th>
<th>city</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>44</td>
<td>131</td>
<td>1234567</td>
<td>Mike</td>
<td>Mayfield</td>
<td>NYC</td>
<td>EH4 8LE</td>
</tr>
<tr>
<td>t₂</td>
<td>44</td>
<td>131</td>
<td>3456789</td>
<td>Rick</td>
<td>Crichton</td>
<td>NYC</td>
<td>EH4 8LE</td>
</tr>
<tr>
<td>t₃</td>
<td>01</td>
<td>908</td>
<td>3456789</td>
<td>Joe</td>
<td>Mtn Ave</td>
<td>NYC</td>
<td>07974</td>
</tr>
</tbody>
</table>
Example: customer relation

One of the central technical problems is how to tell whether the data is dirty or clean

- **Schema**: country code (*CC*), area code (*AC*), phone (*phn*), ...

  \[
  \text{Cust} (\text{CC}: \text{int}, \text{AC}: \text{int}, \text{phn}: \text{int}, \text{name}: \text{string}, \text{street}: \text{string}, \text{city}: \text{string}, \text{zip}: \text{string})
  \]

- **Instance**:

<table>
<thead>
<tr>
<th>CC</th>
<th>AC</th>
<th>phn</th>
<th>name</th>
<th>street</th>
<th>city</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>131</td>
<td>1234567</td>
<td>Mike</td>
<td>Mayfield</td>
<td>NYC</td>
<td>EH4 8LE</td>
</tr>
<tr>
<td>44</td>
<td>131</td>
<td>3456789</td>
<td>Rick</td>
<td>Crichton</td>
<td>NYC</td>
<td>EH4 8LE</td>
</tr>
<tr>
<td>01</td>
<td>908</td>
<td>3456789</td>
<td>Joe</td>
<td>Mtn Ave</td>
<td>NYC</td>
<td>07974</td>
</tr>
</tbody>
</table>

- **Functional dependencies (FDs)**:

  \[
  f_1: [\text{CC}, \text{AC}, \text{phn}] \rightarrow [\text{street}, \text{city}, \text{zip}],
  \]

  \[
  f_2: [\text{CC}, \text{AC}] \rightarrow [\text{city}].
  \]

The database satisfies the FDs. Is the data really clean?
Capturing inconsistencies in the data

- In the **UK**, the zip code uniquely determines the street.
  \[ \text{cfd}_1: (\text{CC} = 44, \text{zip}) \rightarrow \text{[street]}] \]
- This constraint specifies a **semantic** property of the data.
- It does **not** hold for other countries, e.g., USA
- It can’t be expressed as standard FDs.
Capturing inconsistencies in the data

- In the UK, the zip code uniquely determines the street.

  \[ \text{cfd}_1: ([\text{CC} = 44, \text{zip}] \rightarrow [\text{street}]) \]

- This constraint specifies a semantic property of the data.

- It does not hold for other countries, e.g., USA

- It can’t be expressed as standard FDs.

- The example database does not satisfy this constraint

<table>
<thead>
<tr>
<th></th>
<th>CC</th>
<th>AC</th>
<th>phn</th>
<th>name</th>
<th>street</th>
<th>city</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_1):</td>
<td>44</td>
<td>131</td>
<td>1234567</td>
<td>Mike</td>
<td>Mayfield</td>
<td>NYC</td>
<td>EH4 8LE</td>
</tr>
<tr>
<td>(t_2):</td>
<td>44</td>
<td>131</td>
<td>3456789</td>
<td>Rick</td>
<td>Crichton</td>
<td>NYC</td>
<td>EH4 8LE</td>
</tr>
<tr>
<td>(t_3):</td>
<td>01</td>
<td>908</td>
<td>3456789</td>
<td>Joe</td>
<td>Mtn Ave</td>
<td>NYC</td>
<td>07974</td>
</tr>
</tbody>
</table>

The data is not clean after all
More example constraints

- In the UK, if the area code is 131, then the city must be Edinburgh (EDI).
- In the USA, if the area code is 908, then the city must be Murray Hill (MH).
- Refining the FD $f_1$: $[CC, AC, phn] \rightarrow [street, city, zip]$ by adding conditions (bindings of semantically related constants):
  
  $cfd_2$: $[CC = 44, AC = 131, phn] \rightarrow [street, city = 'EDI', zip])$
  
  $cfd_3$: $[CC = 01, AC = 908, phn] \rightarrow [street, city = 'MH', zip])$
More example constraints

- In the UK, if the area code is 131, then the city must be Edinburgh (EDI)
- In the USA, if the area code is 908, then the city must be Murray Hill (MH)
- Refining the FD $f_1: [CC, AC, phn] \rightarrow [street, city, zip]$ by adding conditions (bindings of semantically related constants)

  \[
  \text{cfd}_2: ([CC = 44, AC = 131, phn] \rightarrow [\text{street}, \text{city} = '\text{EDI}', \text{zip}])
  \]
  \[
  \text{cfd}_3: ([CC = 01, AC = 908, phn] \rightarrow [\text{street}, \text{city} = '\text{MH}', \text{zip}])
  \]

<table>
<thead>
<tr>
<th></th>
<th>CC</th>
<th>AC</th>
<th>phn</th>
<th>name</th>
<th>street</th>
<th>city</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>44</td>
<td>131</td>
<td>1234567</td>
<td>Mike</td>
<td>Mayfield</td>
<td>NYC</td>
<td>EH4 8LE</td>
</tr>
<tr>
<td>$t_2$</td>
<td>44</td>
<td>131</td>
<td>3456789</td>
<td>Rick</td>
<td>Crichton</td>
<td>NYC</td>
<td>EH4 8LE</td>
</tr>
<tr>
<td>$t_3$</td>
<td>01</td>
<td>908</td>
<td>3456789</td>
<td>Joe</td>
<td>Mtn Ave</td>
<td>NYC</td>
<td>07974</td>
</tr>
</tbody>
</table>

None of the tuples in the example database is clean
The need for new dependencies

cfd_1: ([CC = 44, zip] → [street])
cfd_2: ([CC = 44, AC = 131, phn] → [street, city = ‘EDI’, zip])
cfd_3: ([CC = 01, AC = 908, phn] → [street, city = ‘MH’, zip])

- They capture inconsistencies that traditional FDs cannot detect – FDs were designed for schema design after all
- Data integration in real-life: source dependencies
  - hold on a subset of sources
  - but only hold conditionally on the integrated data
- They are NOT expressible as traditional FDs
  - do not hold on the entire relation
  - contain constant data values, besides logical variables

To determine whether the data is dirty or clean
Conditional Functional Dependencies (CFDs)

An extension of traditional functional dependencies:

- A CFD is defined to be a pair \( \varphi = R(X \rightarrow Y, T_p) \), where
  - \( X \rightarrow Y \) is a standard FD, embedded in \( \varphi \);
  - \( T_p \) is the pattern tableau consisting of tuples \( t_p \) over \( X \cup Y \);
  - In a pattern tuple \( t_p \), each \( t_p[A] \) is either a constant from \( \text{dom}(A) \) or a wildcard ‘\( _\)' (unnamed variable) that draws values from \( \text{dom}(A) \).

Example: \( \text{cfd}_1: ([CC = 44, zip] \rightarrow [street]) \)

- \( \text{Cust}([CC, zip] \rightarrow [street], T_p) \)
- pattern tableau \( T_p: \)

<table>
<thead>
<tr>
<th>CC</th>
<th>zip</th>
<th>street</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>
Conditional Functional Dependencies (CFDs)

- A CFD is defined to be a pair $\varphi = R(X \rightarrow Y, T_p)$, where
  - $X \rightarrow Y$ is a standard FD, embedded in $\varphi$;
  - $T_p$ is the pattern tableau consisting of tuples $t_p$ over $X \cup Y$;
  - In a pattern tuple $t_p$, each $t_p[A]$ is either a constant from $\text{dom}(A)$ or a wildcard ‘-’ (unnamed variable) that draws values from $\text{dom}(A)$.

- Traditional FDs as a special case: expressing the FD $f_1$ as
  - $\text{Cust}([CC, AC, phn] \rightarrow [street, city, zip], T_p)$
  - pattern tableau $T_p$:

<table>
<thead>
<tr>
<th></th>
<th>CC</th>
<th>AC</th>
<th>phn</th>
<th>street</th>
<th>city</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

CFDs subsume traditional FDs.
A CFD is defined to be a pair $\varphi = R(X \rightarrow Y, T_p)$, where
- $X \rightarrow Y$ is a standard FD, embedded in $\varphi$;
- $T_p$ is the pattern tableau consisting of tuples $t_p$ over $X \cup Y$;
- In a pattern tuple $t_p$, each $t_p[A]$ is either a constant from $\text{dom}(A)$ or a wildcard ‘_’ (unnamed variable) that draws values from $\text{dom}(A)$.

A single CFD representing $\text{cfd}_2$, $\text{cfd}_3$ and FD $f_1$:
- $\text{Cust}([CC, AC, phn] \rightarrow [street, city, zip], T_p)$
- pattern tableau $T_p$:

<table>
<thead>
<tr>
<th></th>
<th>CC</th>
<th>AC</th>
<th>phn</th>
<th>street</th>
<th>city</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>44</td>
<td>131</td>
<td>_</td>
<td>_</td>
<td>EDI</td>
<td>_</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>908</td>
<td>_</td>
<td>_</td>
<td>MH</td>
<td>_</td>
</tr>
<tr>
<td>3</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
</tr>
</tbody>
</table>

Each pattern tuple $t_p$ is a constraint.
Semantics of CFDs

- **Operator \( \simeq \):**
  - a matches b \( (a \simeq b) \)
    - either a or b is \( _\_ \)
    - both a and b are constants and \( a = b \).
  - tuple \( t_1 \) matches tuple \( t_2 \) \( (t_1 \simeq t_2) \): defined component-wise
    - \( (a, b) \simeq (a, _) \) but \( (a, b) \not\simeq (a, c) \).
Semantics of CFDs

- **Operator \( \bowtie \):**
  - *a matches b* \( (a \bowtie b) \)
    - either \( a \) or \( b \) is \( \_ \)
    - both \( a \) and \( b \) are constants and \( a = b \).
  - tuple \( t_1 \) matches tuple \( t_2 \) \( (t_1 \bowtie t_2) \): defined component-wise
    - \((a, b) \bowtie (a, \_)\) but \((a, b) \not\bowtie (a, c)\).

- A database \( D \) satisfies a CFD \( \varphi = R(X \rightarrow Y, T_p) \) iff for each pair of tuples \( u, v \in D \) and for each pattern tuple \( t_p \in T_p \),
  - if \( u[X] = v[X] \bowtie t_p[X] \), then \( u[Y] = v[Y] \bowtie t_p[Y] \)
Semantics of CFDs

- Operator \( \simeq \):
  - \( a \) matches \( b \) (\( a \simeq b \))
    - either \( a \) or \( b \) is _
    - both \( a \) and \( b \) are constants and \( a = b \).
  - tuple \( t_1 \) matches tuple \( t_2 \) (\( t_1 \simeq t_2 \)): defined component-wise
    - \((a, b) \simeq (a, _)\) but \((a, b) \not\approx (a, c)\).

- A database \( D \) satisfies a CFD \( \varphi = R(X \rightarrow Y, T_p) \) iff for each pair of tuples \( u, v \in D \) and for each pattern tuple \( t_p \in T_p \),
  
  if \( u[X] = v[X] \simeq t_p[X] \), then \( u[Y] = v[Y] \simeq t_p[Y] \)

- Pattern tuples:
  - \( t_p[X] \): identifying a subset \( \{ u \mid u \in D, u[X] \simeq t_p[X] \} \);
  - \( u[Y] = v[Y] \simeq t_p[Y] \): enforcing the FD \( X \rightarrow Y \) and the pattern \( t_p[Y] \) to the subset.

**Conditional**: \( t_p \) applies to the subset rather than to the entire \( D \)
Violation of CFDs

- Cust([CC, AC, phn] → [street, city, zip], Tp)

<table>
<thead>
<tr>
<th>Tp:</th>
<th>CC</th>
<th>AC</th>
<th>phn</th>
<th>street</th>
<th>city</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>44</td>
<td>131</td>
<td></td>
<td></td>
<td>EDI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>908</td>
<td></td>
<td></td>
<td>MH</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Tuple t3 violates the CFD:
  - t3[CC, AC, phn] = t3[CC, AC, phn] ≃ t p[CC, AC, phn]
  - t3[city] ≠ t p[city]

<table>
<thead>
<tr>
<th></th>
<th>CC</th>
<th>AC</th>
<th>phn</th>
<th>name</th>
<th>street</th>
<th>city</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1:</td>
<td>44</td>
<td>131</td>
<td>1234567</td>
<td>Mike</td>
<td>Mayfield</td>
<td>NYC</td>
<td>EH4 8LE</td>
</tr>
<tr>
<td>t2:</td>
<td>44</td>
<td>131</td>
<td>3456789</td>
<td>Rick</td>
<td>Crichton</td>
<td>NYC</td>
<td>EH4 8LE</td>
</tr>
<tr>
<td>t3:</td>
<td>01</td>
<td>908</td>
<td>3456789</td>
<td>Joe</td>
<td>Mtn Ave</td>
<td>NYC</td>
<td>07974</td>
</tr>
</tbody>
</table>
Violation of CFDs

- Cust([CC, AC, phn] → [street, city, zip], Tp)

<table>
<thead>
<tr>
<th></th>
<th>CC</th>
<th>AC</th>
<th>phn</th>
<th>street</th>
<th>city</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tp:</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>44</td>
<td>131</td>
<td>-</td>
<td>-</td>
<td>EDI</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>908</td>
<td>_</td>
<td>_</td>
<td>MH</td>
<td>_</td>
</tr>
</tbody>
</table>

- Tuple t3 violates the CFD:
  - t3[CC, AC, phn] = t3[CC, AC, phn] ≍ t_p[CC, AC, phn]
  - t3[city] ≠ t_p[city]

<table>
<thead>
<tr>
<th></th>
<th>CC</th>
<th>AC</th>
<th>phn</th>
<th>name</th>
<th>street</th>
<th>city</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1:</td>
<td>44</td>
<td>131</td>
<td>1234567</td>
<td>Mike</td>
<td>Mayfield</td>
<td>NYC</td>
<td>EH4 8LE</td>
</tr>
<tr>
<td>t2:</td>
<td>44</td>
<td>131</td>
<td>3456789</td>
<td>Rick</td>
<td>Crichton</td>
<td>NYC</td>
<td>EH4 8LE</td>
</tr>
<tr>
<td>t3:</td>
<td>01</td>
<td>908</td>
<td>3456789</td>
<td>Joe</td>
<td>Mtn Ave</td>
<td>NYC</td>
<td>07974</td>
</tr>
</tbody>
</table>

A single tuple may violate a CFD
The need for extending inclusion dependencies

Example schema:

**Source:** order(title: string, type: string, price: real)

**Target:** book(title: string, price: real, format: string)

CD(album: string, price: real, genre: string)

Inclusion dependencies (INDs) from the source to the target?
The need for extending inclusion dependencies

Example schema:

**Source:** \( \text{order}(\text{title}: \text{string}, \text{type}: \text{string}, \text{price}: \text{real}) \)

**Target:** \( \text{book}(\text{title}: \text{string}, \text{price}: \text{real}, \text{format}: \text{string}) \)

\( \text{CD}(\text{album}: \text{string}, \text{price}: \text{real}, \text{genre}: \text{string}) \)

Inclusion dependencies (INDs) from the source to the target?

\[
\text{order}(\text{title}, \text{price}) \subseteq \text{book}(\text{title}, \text{price}), \\
\text{order}(\text{title}, \text{price}) \subseteq \text{CD}(\text{album}, \text{price}).
\]

These traditional INDs do not make sense

<table>
<thead>
<tr>
<th>order</th>
<th>title</th>
<th>type</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_4):</td>
<td>Snow White</td>
<td>CD</td>
<td>7.99</td>
</tr>
<tr>
<td>(t_5):</td>
<td>Harry Potter</td>
<td>book</td>
<td>17.99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>book</th>
<th>title</th>
<th>price</th>
<th>format</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_6):</td>
<td>Harry Potter</td>
<td>17.99</td>
<td>hard-cover</td>
</tr>
<tr>
<td>(t_7):</td>
<td>Snow White</td>
<td>7.99</td>
<td>paper-cover</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CD</th>
<th>album</th>
<th>price</th>
<th>genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_8):</td>
<td>J. Denver</td>
<td>7.94</td>
<td>country</td>
</tr>
<tr>
<td>(t_9):</td>
<td>Snow White</td>
<td>7.99</td>
<td>a-book</td>
</tr>
</tbody>
</table>
Extending inclusion dependencies for schema matching

◮ Schema:

**Source:** `order(title: string, type: string, price: real)`

**Target:** `book(title: string, price: real, format: string)`

`CD(album: string, price: real, genre: string)`

◮ There are indeed inclusion dependencies, under conditions:

\[
cind_1: (\text{order}(\text{title, price}; \text{type} = \text{'book'}) \subseteq \text{book}(\text{title, price}))
\]

\[
cind_2: (\text{order}(\text{title, price}; \text{type} = \text{'CD'}) \subseteq \text{CD}(\text{album, price}))
\]

◮ `order(title, price) \subseteq \text{book}(title, price)` holds only if `type = \text{book}

◮ `order(title, price) \subseteq \text{CD}(\text{album, price})` holds only if `type = \text{CD}

These dependencies only apply to subsets of the order relation that satisfy certain patterns.
Extending inclusion dependencies for data cleaning

- A constraint from CD to book: it holds only if the genre of a CD is audio book and if so, then the format of the matching book must be audio

\[ \text{cind}_3: \ (\text{CD}(\text{album}, \text{price}; \ \text{genre} = \text{‘a-book’}) \subseteq \text{book}(\text{title}, \text{price}; \ \text{format} = \text{‘audio’})) \]

- The example database does not satisfy \( \text{cind}_3 \)

<table>
<thead>
<tr>
<th>CD</th>
<th>album</th>
<th>price</th>
<th>genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_8 ):</td>
<td>J. Denver</td>
<td>7.94</td>
<td>country</td>
</tr>
<tr>
<td>( t_9 ):</td>
<td>Snow White</td>
<td>7.99</td>
<td>a-book</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>book</th>
<th>title</th>
<th>price</th>
<th>format</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_6 ):</td>
<td>Harry Potter</td>
<td>17.99</td>
<td>hard-cover</td>
</tr>
<tr>
<td>( t_7 ):</td>
<td>Snow White</td>
<td>7.99</td>
<td>paper-cover</td>
</tr>
</tbody>
</table>
Extending inclusion dependencies for data cleaning

- A constraint from CD to book: it holds only if the genre of a CD is audio book and if so, then the format of the matching book must be audio

\[
cind_3: (\text{CD(album, price; genre = 'a-book')} \subseteq \text{book(title, price; format = 'audio')})
\]

- The example database does not satisfy \(cind_3\)

<table>
<thead>
<tr>
<th>CD</th>
<th>album</th>
<th>price</th>
<th>genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_8):</td>
<td>J. Denver</td>
<td>7.94</td>
<td>country</td>
</tr>
<tr>
<td>(t_9):</td>
<td>Snow White</td>
<td>7.99</td>
<td>a-book</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>book</th>
<th>title</th>
<th>price</th>
<th>format</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_6):</td>
<td>Harry Potter</td>
<td>17.99</td>
<td>hard-cover</td>
</tr>
<tr>
<td>(t_7):</td>
<td>Snow White</td>
<td>7.99</td>
<td>paper-cover</td>
</tr>
</tbody>
</table>

These dependencies specify patterns of semantically related data values across different relations
An extension of inclusion dependencies:

- A CIND is a pair \((R_1[X] \subseteq R_2[Y], \ T_p[X_p \parallel Y_p])\), where
  - \(R_1[X] \subseteq R_2[Y]\) is a standard IND from \(R_1\) to \(R_2\);
  - \(T_p\) is a pattern tableau over \(X_p\) of \(R_1\) and \(Y_p\) of \(R_2\) (distinct from \(X\) and \(Y\)), consisting of pattern tuples of constants only.
Conditional Inclusion Dependencies (CINDs)

- A CIND is a pair $(R_1[X] \subseteq R_2[Y], \ T_p[X_p \parallel Y_p])$, where
  - $R_1[X] \subseteq R_2[Y]$ is a standard IND from $R_1$ to $R_2$;
  - $T_p$ is a pattern tableau over $X_p$ of $R_1$ and $Y_p$ of $R_2$ (distinct from $X$ and $Y$), consisting of pattern tuples of constants only.

- Examples: $\text{cind}_1$, $\text{cind}_2$, $\text{cind}_3$:
  - $\text{cind}_1$: $(\text{order}(\text{title, price}; \text{type} = \text{'book'}) \subseteq \text{book}(\text{title, price}))$
  - $\text{cind}_2$: $(\text{order}(\text{title, price}; \text{type} = \text{'CD'}) \subseteq \text{CD}(\text{album, price}))$
  - $\text{cind}_3$: $(\text{CD}(\text{album, price}; \text{genre} = \text{'a-book'}) \subseteq \text{book}(\text{title, price}; \text{format} = \text{'audio'}))$

- CINDs:
  - $\varphi_4$: $(\text{order}(\text{title, price}) \subseteq \text{book}(\text{title, price}), \ T_4[\text{type}])$
  - $\varphi_5$: $(\text{order}(\text{title, price}) \subseteq \text{CD}(\text{album, price}), \ T_5[\text{type}])$
  - $\varphi_6$: $(\text{CD}(\text{album, price}) \subseteq \text{book}(\text{title, price}), \ T_6[\text{genre} \parallel \text{format}])$

$$
\begin{array}{c|c}
\text{type} & \text{book} \\
\hline
\text{type} & \text{CD} \\
\hline
\text{genre} & \text{format} \\
\hline
\text{a-book} & \text{audio}
\end{array}
$$
A CIND is a pair \((R_1[X] \subseteq R_2[Y], \ T_p[X_p \parallel Y_p])\), where
- \(R_1[X] \subseteq R_2[Y]\) is a standard IND from \(R_1\) to \(R_2\);
- \(T_p\) is a pattern tableau over \(X_p\) of \(R_1\) and \(Y_p\) of \(R_2\) (distinct from \(X\) and \(Y\)), consisting of pattern tuples of constants only.

Standard INDs are a special case of CINDs:
- IND: \(R_1[X] \subseteq R_2[Y]\)
- CIND: \((R_1[X] \subseteq R_2[Y], \ T_p[\emptyset])\)
Semantics of CINDs

- \( D = (D_1, D_2), \) where \( D_i \) is an instance of \( R_i, \ i = 1, 2. \)
- \( D \) satisfies \((R_1[X] \subseteq R_2[Y], T_p[X_p \parallel Y_p])\) iff
  for any tuple \( s \) in \( D_1 \) and any pattern tuple \( t_p \) in \( T_p, \)
  if \( s[X_p] = t_p[X_p] \) then there exists a tuple \( t \) in \( D_2 \) such that
  - \( s[X] = t[Y] \) and
  - \( t[Y_p] = t_p[Y_p]. \)
Semantics of CINDs

$D = (D_1, D_2)$, where $D_i$ is an instance of $R_i$, $i = 1, 2$.

$D$ satisfies $(R_1[X] \subseteq R_2[Y], T_p[X_p \parallel Y_p])$ iff for any tuple $s$ in $D_1$ and any pattern tuple $t_p$ in $T_p$ if $s[X_p] = t_p[X_p]$ then there exists a tuple $t$ in $D_2$ such that

- $s[X] = t[Y]$ and

Pattern tuples:

- $t_p[X_p]$ identifies a subset $\{s \mid s \in D_1, s[X_p] = t_p[X_p]\}$;
- $s[X] = t[Y]$ and $t[Y_p] = t_p[Y_p]$: enforcing the standard IND $R_1[X] \subseteq R_2[Y]$ on the subset, and moreover, enforcing the $t_p[Y_p]$ pattern to the matching $R_2$ tuples.

Each pattern tuple $t_p$ is a constraint
Other extensions: Denial constraints

- Well studied for improving data quality
- Universally quantified FO sentences of the form:

\[ \forall \bar{x}_1 \ldots \bar{x}_m \neg (R_1(\bar{x}_1) \land \ldots \land R_m(\bar{x}_m) \land \varphi(\bar{x}_1, \ldots, \bar{x}_m)), \]

- \( R_i \) is a relation atom for \( i \in [1, m] \);
- \( \varphi \) is a conjunction of built-in predicates such as \( =, \neq, <, >, \leq, \geq \);
- may carry constants, numerical values and aggregate functions.

- Static analyses: satisfiability, implication and finite axiomatizability?

Surveys:

Other extensions of functional dependencies

- Studied for constraint databases and constraint logic programs
- **Constraint Generating Dependencies:**
  \[ \forall \bar{x}(R_1(\bar{x}) \land \ldots \land R_k(\bar{x}) \land \xi(\bar{x}) \rightarrow \xi'(\bar{x})) \]
  - \(\xi, \xi'\) are arbitrary constraints, which may carry constants;
  - subsuming CFDs

Other extensions of functional dependencies

- Studied for constraint databases and constraint logic programs

**Constraint Generating Dependencies:**

\[ \forall \vec{x}(R_1(\vec{x}) \land \ldots \land R_k(\vec{x}) \land \xi(\vec{x}) \rightarrow \xi'(\vec{x})) \]

- \( \xi, \xi' \) are arbitrary constraints, which may carry constants;
- subsuming CFDs


**Constrained Tuple Generating Dependencies:**

\[ \forall \vec{x}(R_1(\vec{x}) \land \ldots \land R_k(\vec{x}) \land \xi \rightarrow \exists \vec{y}(R_1'(\vec{x}, \vec{y}) \land \ldots \land R_s'(\vec{x}, \vec{y}) \land \xi' (\vec{x}, \vec{y}))) \]

subsuming both CFDs and CINDs;

Other extensions of functional dependencies

- Studied for constraint databases and constraint logic programs

- **Constraint Generating Dependencies:**

\[
\forall \bar{x}(R_1(\bar{x}) \land \ldots \land R_k(\bar{x}) \land \xi(\bar{x}) \rightarrow \xi'(\bar{x}))
\]

  - \(\xi, \xi'\) are arbitrary constraints, which may carry constants;
  - subsuming CFDs


- **Constrained Tuple Generating Dependencies:**

\[
\forall \bar{x}(R_1(\bar{x}) \land \ldots \land R_k(\bar{x}) \land \xi \rightarrow \exists \bar{y}(R'_1(\bar{x}, \bar{y}) \land \ldots \land R'_s(\bar{x}, \bar{y}) \land \xi'(\bar{x}, \bar{y})))
\]

  subsuming both CFDs and CINDs;


- Higher complexity for static analyses: satisfiability, implication and finite axiomatizability
Survey on traditional data dependencies:


Extensions of traditional dependencies:


**CFDs**


**CINDs**

Extending dependencies with conditions for capturing data inconsistencies
  - Conditional functional dependencies (CFDs)
  - Conditional inclusion dependencies (CINDs)
  - Other extensions
Matching dependencies for object identification
  - Object identification and matching rules
  - Matching dependencies
Static analyses: New challenges
Improving data quality with dependencies
Open research issues
Object identification

Data deduplication, merge/purge, record linkage (matching): to identify tuples from one or more relations that refer to the same real-world object.
Object identification

Data deduplication, merge/purge, record linkage (matching): to identify tuples from one or more relations that refer to the same real-world object.

Example: credit-card fraud detection

➤ Schema: credit cards and billing transactions
  card(C#, type, SSN, FN, LN, addr, tel, email),
  billing(C#, item, price, FN, SN, post, phn, email).

➤ For any instance \((D_c, D_b)\) of \((\text{card}, \text{billing})\), \(t \in D_c, t' \in D_b\),
  ➤ if \(t[C\#] = t'[C\#]\),
  ➤ then \(t[Y_c]\) and \(t'[Y_b]\) must match – refer to the same holder
  \(Y_c = [\text{FN, LN, addr, tel, email}], \quad Y_b = [\text{FN, SN, post, phn, email}].\)
Object identification

Data deduplication, merge/purge, record linkage (matching): to identify tuples from one or more relations that refer to the same real-world object.

Example: credit-card fraud detection

- Schema: credit cards and billing transactions
  - card(C#, type, SSN, FN, LN, addr, tel, email),
  - billing(C#, item, price, FN, SN, post, phn, email).
- For any instance \((D_c, D_b)\) of (card, billing), \(t \in D_c, t' \in D_b\),
  - if \(t[C#] = t'[C#]\),
  - then \(t[Y_c]\) and \(t'[Y_b]\) must match – refer to the same holder

\[Y_c = [FN, LN, addr, tel, email], \quad Y_b = [FN, SN, post, phn, email].\]

essential to data integration, data cleaning, ...
Matching rules

Challenges: unreliable data sources, different representations ...

Matching rules: what attributes to compare and how to compare the attributes

- if $t[\text{LN, addr}]$ and $t'[\text{SN, post}]$ match, and
- if $t[\text{FN}]$ and $t'[\text{FN}]$ either match or are similar w.r.t. a similarity relation $\approx_d$,
- then $t[\text{Y}_c]$ and $t'[\text{Y}_b]$ match
Matching rules

Challenges: unreliable data sources, different representations ...

Matching rules: what attributes to compare and how to compare the attributes

- if \( t[\text{LN, addr}] \) and \( t'[\text{SN, post}] \) match, and
- if \( t[\text{FN}] \) and \( t'[\text{FN}] \) either match or are similar w.r.t. a similarity relation \( \approx_d \),
- then \( t[\text{Y}_c] \) and \( t'[\text{Y}_b] \) match

We can identify \( t[\text{Y}_c] \) and \( t'[\text{Y}_b] \) even if they radically differ in some attributes

- comparing \( t[\text{LN, addr, FN}] \) and \( t'[\text{SN, post, FN}] \) instead of \( t[\text{FN, LN, addr, tel, email}] \) and \( t'[\text{FN, SN, post, phn, email}] \).
- similarity \( \approx_d \) in stead of equality on \( \text{FN} \)
Expressing matching rules as dependencies

Match relation: $\equiv$

- if $t[\text{LN}, \text{addr}] \equiv t'[\text{SN}, \text{post}]$, and
- if either $t[\text{FN}] \equiv t'[\text{FN}]$ or $t[\text{FN}] \approx d t'[\text{FN}]$,
- then $t[\text{Y}_c] \equiv t'[\text{Y}_b]$

$\phi_1$: $	ext{card}[\text{LN}] \equiv \text{billing}[\text{SN}] \land \text{card}[\text{addr}] \equiv \text{billing}[\text{post}] \land$
$\text{card}[\text{FN}] \equiv \text{billing}[\text{FN}] \rightarrow \text{card}[\text{Y}_c] \equiv \text{billing}[\text{Y}_b]$

$\phi_2$: $	ext{card}[\text{LN}] \equiv \text{billing}[\text{SN}] \land \text{card}[\text{addr}] \equiv \text{billing}[\text{post}] \land$
$\text{card}[\text{FN}] \approx d \text{billing}[\text{FN}] \rightarrow \text{card}[\text{Y}_c] \equiv \text{billing}[\text{Y}_b]$
Domain specific relations

A fixed set $\Theta$ of similarity relations: for each $\approx$ in $\Theta$,

- **reflexive**: $x \approx x$;
- **symmetric**: if $x \approx y$ then $y \approx x$;
- **subsuming equality** $\equiv$: if $x = y$ then $x \approx y$. 

Wenfei Fan – Dependencies Revisited for Improving Data Quality 28 / 70
Domain specific relations

A fixed set $\Theta$ of similarity relations: for each $\approx$ in $\Theta$,
- **reflexive**: $x \approx x$;
- **symmetric**: if $x \approx y$ then $y \approx x$;
- **subsuming equality** $=$: if $x = y$ then $x \approx y$.

Special relations in $\Theta$:
- **equality** $=$
Domain specific relations

A fixed set $\Theta$ of similarity relations: for each $\approx$ in $\Theta$,

- **reflexive**: $x \approx x$;
- **symmetric**: if $x \approx y$ then $y \approx x$;
- **subsuming equality** $\equiv$: if $x = y$ then $x \approx y$.

Special relations in $\Theta$:

- **equality** $\equiv$
- **match relation** $\Leftrightarrow$ defined on value lists:
  - **transitivity**: if $L_1 \Leftrightarrow L_2$ and $L_2 \Leftrightarrow L_3$ then $L_1 \Leftrightarrow L_3$.
  - **pairwise match**: for $L = [L_1, \ldots, L_k]$ and $L' = [L'_1, \ldots, L'_k]$,
    
    $L \Leftrightarrow L'$ iff $L_j \Leftrightarrow L_j$ for all $j \in [1, k]$.

Domain specific: $\approx$ may not be expressible in FO
Matching dependencies (MDs)

- An MD $\phi$ defined on schemas $(R_1, R_2)$:

$$\bigwedge_{j \in [1,k]} (R_1[X_1[j]] \approx_j R_2[X_2[j]]) \rightarrow R_1[Z_1] \approx R_2[Z_2]$$

- $\approx$ and $\approx_j$ are similarity relations in $\Theta$;
- $X_1, X_2$ (resp. $Z_1, Z_2$): attribute lists of $R_1, R_2$

- The MD $\phi$ holds on $(D_1, D_2)$, where $D_i$ is an instance of $R_i$, iff for any tuple $u$ in $D_1$ and any tuple $v$ in $D_2$, if $\bigwedge_{j \in [1,k]} u[X_1[j]] \approx_j v[X_2[j]]$, then $u[Z_1] \approx v[Z_2]$.

$\phi_1$: $\text{card[LN]} \Leftarrow \text{billing[SN]} \land \text{card[addr]} \Leftarrow \text{billing[post]} \land \text{card[FN]} \Leftarrow \text{billing[FN]} \rightarrow \text{card[Y_c]} \Leftarrow \text{billing[Y_b]}$

$\phi_2$: $\text{card[LN]} \Leftarrow \text{billing[SN]} \land \text{card[addr]} \Leftarrow \text{billing[post]} \land \text{card[FN]} \approx_d \text{billing[FN]} \rightarrow \text{card[Y_c]} \Leftarrow \text{billing[Y_b]}$
Example matching dependencies

- If \( t[tel] \) and \( t'[phn] \) equal, then \( t[addr] \Leftrightarrow t'[post] \)
- If \( t[email] \) and \( t'[email] \) equal, then \( t[FN, LN] \Leftrightarrow t'[FN, SN] \).

\( \phi_3: \) \( \text{card}[tel] = \text{billing}[phn] \rightarrow \text{card}[addr] \Leftrightarrow \text{billing}[post] \)

\( \phi_4: \) \( \text{card}[email] = \text{billing}[email] \rightarrow \text{card}[FN, LN] \Leftrightarrow \text{billing}[FN, SN] \)
Known vs. unknown relations

\[
\text{card}[\text{LN}] \iff \text{billing}[\text{SN}] \land \text{card}[\text{addr}] \iff \text{billing}[\text{post}] \land \\
\text{card}[\text{FN}] \approx_d \text{billing}[\text{FN}] \rightarrow \text{card}[Y_c] \iff \text{billing}[Y_b]
\]

- **Similarity** \(\approx\) (except \(\iff\)), e.g., =, \(\approx_d\): to compare data values in unreliable sources
  - similarity metrics: edit distance, \(q\)-grams, Jaro distance, ...
  - total mappings defined on specific domains, already given
Known vs. unknown relations

card[FN] ≈_d billing[FN] → card[Y_c] ⇔ billing[Y_b]

- **Similarity**: \( \approx (\text{except } \rightleftharpoons) \), e.g., =, \( \approx_d \): to compare data values in unreliable sources
  - similarity metrics: edit distance, \( q \)-grams, Jaro distance, ...
  - total mappings defined on specific domains, already given

- **Match relation**: \( \rightleftharpoons \):
  - either not given or partially defined;
  - to be “inferred” via generic reasoning about matching rules;
  - \( u[Z_1] \rightleftharpoons v[Z_2] \)
    - \( u[Z_1] \) and \( v[Z_2] \) refer to the same object;
    - \( u[Z_1] \) and \( v[Z_2] \) may not be directly matched using any metric
      \( \approx \) known in advance.
Known vs. unknown relations

card[FN] ≈ₜ billing[FN] → card[Y_c] ⇔ billing[Y_b]

- **Similarity** ≈ (except ⇔), e.g., =, ≈ₜ: to compare data values in unreliable sources
  - similarity metrics: edit distance, q-grams, Jaro distance, ...
  - total mappings defined on specific domains, already given

- **Match relation** ⇔:
  - either not given or partially defined;
  - to be “inferred” via generic reasoning about matching rules;
  - \( u[Z_1] ⇔ v[Z_2] \)
    - \( u[Z_1] \) and \( v[Z_2] \) refer to the same object;
    - \( u[Z_1] \) and \( v[Z_2] \) may not be directly matched using any metric ≈ known in advance.

- **Matching dependencies**: essentially used to infer the match relation ⇔ (implication analysis)
Matching dependencies vs. Functional dependencies

An extension of traditional functional dependencies (FDs)

- **MDs**: \( \bigwedge_{j \in [1,k]}(R_1[X_1[j]] \approx_j R_2[X_2[j]]) \rightarrow R_1[Z_1] \approx R_2[Z_2] \)

- **FDs** \( R(X \rightarrow Y) \): a special form of MDs when \( R_1 \) and \( R_2 \) are both \( R \), \( X_1[j] \) and \( X_2[j] \) are the same attribute for \( j \in [1,k] \), \( Z_1 \) and \( Z_2 \) are the same attribute list, and \( \approx_j \) and \( \approx \) are =.
Matching dependencies vs. Functional dependencies

An extension of traditional functional dependencies (FDs)

- **MDs**: \( \bigwedge_{j \in [1,k]} (R_1[X_1[j]] \approx_j R_2[X_2[j]]) \rightarrow R_1[Z_1] \approx R_2[Z_2] \)

- **FDs** \( R(X \rightarrow Y) \): a special form of MDs when \( R_1 \) and \( R_2 \) are both \( R \), \( X_1[j] \) and \( X_2[j] \) are the same attribute for \( j \in [1,k] \), \( Z_1 \) and \( Z_2 \) are the same attribute list, and \( \approx_j \) and \( \approx \) are =.

Differences:

- MDs may be defined **across different relations**, while FDs on a **single** relation
- MDs may be defined in terms of **similarity**, while FDs with **equality** only
- Implication analysis of MDs is **quite different** from its FD counterpart – coming up shortly
References


Outline

- Conditional dependencies for capturing data inconsistencies
  - Conditional functional dependencies (CFDs)
  - Conditional inclusion dependencies (CINDs)
  - Other extensions

- Matching dependencies for object identification
  - Object identification and matching rules
  - Matching dependencies

- Static analyses: New challenges
  - Reasoning about conditional dependencies: Satisfiability, implication, axiomatizability, dependency propagation
  - Inferring matching rules

- Improving data quality with dependencies
  - Data repairing (Arenas, Bertossi, Chomicki)
  - Consistent querying answering (Arenas, Bertossi, Chomicki)
  - Condensed representations of all repairs

- Open research issues
Classical decision problems

- **The satisfiability problem** is to determine, given a schema $\mathcal{R}$ and a set $\Sigma$ of dependencies defined on $\mathcal{R}$, whether or not there exists a **nonempty** database instance $D$ of $\mathcal{R}$ that satisfies all dependencies $\varphi$ in $\Sigma$.

To decide whether or not dependencies are **dirty** themselves

- **The implication problem** is to determine, given a schema $\mathcal{R}$, a set $\Sigma$ of dependencies and a single dependency $\phi$ defined on $\mathcal{R}$, whether or not $\Sigma$ **implies** $\phi$, denoted by $\Sigma \models \phi$, i.e., whether for any each instance $D$ of $\mathcal{R}$ that satisfies $\Sigma$, $D$ also satisfies $\phi$.

To remove redundant dependencies
Reasoning about conditional functional dependencies

For traditional FDs,

- the satisfiability problem is not an issue, and
- the implication problem is in linear time
For traditional FDs,

- the satisfiability problem is not an issue, and
- the implication problem is in linear time

In contrast, a set of CFDs may have conflicts or inconsistencies:

\[ \varphi = R(A \rightarrow B, T_p) \]

- For any nonempty database \( D \) and for any tuple \( t \) in \( D \), \( \varphi \) says that \( t[B] \) must be both \( b_1 \) and \( b_2 \).
In the same setting as the classical dependency theory

Recall domain specification in a schema:

\[
\text{Cust}(\text{CC}: \text{int}, \text{AC}: \text{int}, \text{phn}: \text{int}, \text{name}: \text{string}, \text{street}: \text{string}, \ldots)
\]

It is typically assumed that in each domain,

- there are at least two elements,
- there is no upper bound: possibly infinitely many
In the same setting as the classical dependency theory

Recall domain specification in a schema:

\[
\text{Cust}(\text{CC}: \text{int}, \text{AC}: \text{int}, \text{phn}: \text{int}, \text{name}: \text{string}, \text{street}: \text{string}, \ldots)
\]

It is typically assumed that in each domain,

- there are at least two elements,
- there is no upper bound: possibly infinitely many

**Good news:** in this setting, CFDs do **not** make our lives much harder

**Theorem**

*For CFDs, the satisfiability problem and the implication problem are both in quadratic time.*
In the same setting as the classical dependency theory

Recall domain specification in a schema:

\[
\text{Cust}(CC: \text{int}, AC: \text{int}, phn: \text{int}, name: \text{string}, street: \text{string}, \ldots)
\]

It is typically assumed that in each domain,

- there are at least two elements,
- there is no upper bound: possibly infinitely many

**Good news:** in this setting, CFDs do **not** make our lives much harder

**Theorem**

*For CFDs, the satisfiability problem and the implication problem are both in quadratic time.*

**Bad news:** this no longer holds in the presence of attributes with a finite domain
The interaction between CFDs and domain constraints

In practice, it is common to find attributes with a finite domain: Boolean, date, ...

While the presence of attributes with a finite domain does not complicate the analyses of FDs, it does take a toll on CFDs.

Consider $\Sigma = \{\psi_1, \psi_2\}$, where

$\psi_1 = R(A \rightarrow B, T_1)$, and $\psi_2 = R(B \rightarrow A, T_2)$

If $\text{dom}(A)$ is Boolean, then $\Sigma$ is not satisfiable!
The interaction between CFDs and domain constraints

In practice, it is common to find attributes with a finite domain: Boolean, date, ...

While the presence of attributes with a finite domain does not complicate the analyses of FDs, it does take a toll on CFDs

Consider $\Sigma = \{\psi_1, \psi_2\}$, where

$\psi_1 = R(A \rightarrow B, T_1)$, and $\psi_2 = R(B \rightarrow A, T_2)$

$T_1 = \begin{array}{|c|c|}
\hline
A & B \\
\hline
true & b_1 \\
false & b_2 \\
\hline
\end{array}$

$T_2 = \begin{array}{|c|c|}
\hline
B & A \\
\hline
b_1 & false \\
b_2 & true \\
\hline
\end{array}$

If dom(A) is Boolean, then $\Sigma$ is not satisfiable!
The interaction between CFDs and domain constraints

In practice, it is common to find attributes with a finite domain: Boolean, date, ...

While the presence of attributes with a finite domain does not complicate the analyses of FDs, it does take a toll on CFDs

Consider $\Sigma = \{\psi_1, \psi_2\}$, where

$\psi_1 = R(A \rightarrow B, T_1)$, and $\psi_2 = R(B \rightarrow A, T_2)$

If dom($A$) is Boolean, then $\Sigma$ is not satisfiable!
The interaction between CFDs and domain constraints

In practice, it is common to find attributes with a finite domain: Boolean, date, ...

While the presence of attributes with a finite domain does not complicate the analyses of FDs, it does take a toll on CFDs

Consider $\Sigma = \{\psi_1, \psi_2\}$, where

$\psi_1 = R(A \rightarrow B, T_1)$, and $\psi_2 = R(B \rightarrow A, T_2)$

<table>
<thead>
<tr>
<th></th>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>true</td>
<td>$b_1$</td>
</tr>
<tr>
<td></td>
<td>false</td>
<td>$b_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$B$</th>
<th>$A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_2$</td>
<td>$b_1$</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>$b_2$</td>
<td>true</td>
</tr>
</tbody>
</table>

If $\text{dom}(A)$ is Boolean, then $\Sigma$ is not satisfiable!

**Theorem**

*When attributes with a finite domain may be present,*

- the satisfiability problem for CFDs is \textit{NP-complete}, and
- the implication problem for CFDs is \textit{coNP-complete}.*
Finite axiomatizability of CFDs

Armstrong’s axioms for FDs:

- **Reflexivity**: If $Y \subseteq X$, then $X \rightarrow Y$
- **Augmentation**: If $X \rightarrow Y$, then $XZ \rightarrow YZ$
- **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

**Sound and complete**: $\Sigma \models \phi$ iff $\phi$ can be inferred from $\Sigma$ using the axioms.
Finite axiomatizability of CFDs

Armstrong’s axioms for FDs:

- **Reflexivity**: If $Y \subseteq X$, then $X \rightarrow Y$
- **Augmentation**: If $X \rightarrow Y$, then $XZ \rightarrow YZ$
- **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

**Sound and complete**: $\Sigma \models \phi$ iff $\phi$ can be inferred from $\Sigma$ using the axioms.

**Theorem**

There is a sound and complete inference system for CFDs.
Finite axiomatizability of CFDs

Armstrong’s axioms for FDs:

- Reflexivity: If \( Y \subseteq X \), then \( X \rightarrow Y \)
- Augmentation: If \( X \rightarrow Y \), then \( XZ \rightarrow YZ \)
- Transitivity: If \( X \rightarrow Y \) and \( Y \rightarrow Z \), then \( X \rightarrow Z \)

**Sound and complete:** \( \Sigma \models \phi \) iff \( \phi \) can be inferred from \( \Sigma \) using the axioms.

**Theorem**

There is a sound and complete inference system for CFDs.

More involved than Armstrong’s axioms:

- If \( (X \rightarrow Y, t_p) \) and \( (Y \rightarrow Z, t'_p) \), and
- and if \( t_p[Y] \preceq t'_p[Y] \) (\( \preceq \), \( \preceq \), \( \preceq \)),
- then \( (X \rightarrow Z, (t_p[X] \parallel t'_p[Z])) \)
Static Analyses: CFDs vs. FDs

- In the absence of attributes with a finite domain:

<table>
<thead>
<tr>
<th></th>
<th>satisfiability</th>
<th>implication</th>
<th>finite axiomatizability</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFD</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>yes</td>
</tr>
<tr>
<td>FD</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>yes</td>
</tr>
</tbody>
</table>

- General setting:

<table>
<thead>
<tr>
<th></th>
<th>satisfiability</th>
<th>implication</th>
<th>finite axiomatizability</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFD</td>
<td>NP-complete</td>
<td>coNP-complete</td>
<td>yes</td>
</tr>
<tr>
<td>FD</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>yes</td>
</tr>
</tbody>
</table>

The interaction between domain constraints and CFDs.
Reasoning about conditional inclusion dependencies: Satisfiability

Flashback:
- The satisfiability problem for CFDs in NP-complete in the general setting.
- One can specify any INDs without worrying about their satisfiability.
Reasoning about conditional inclusion dependencies: Satisfiability

Flashback:

- The satisfiability problem for CFDs in NP-complete in the general setting.
- One can specify any INDs without worrying about their satisfiability.

In contrast to CFDs,

**Theorem**

*In the general setting, any set of CINDs is satisfiable.*
The implication problem for traditional INDs is \textit{PSPACE-complete}. 

\textbf{Good news}: the complexity does not hike up in the absence of attributes with a finite domain.

\begin{tcolorbox}[ams, colback=blue!25, awareness]  
\textbf{Theorem} 

\textit{In the absence of attributes with a finite domain, the implication problem for CINDs is \textit{PSPACE-complete}.} 
\end{tcolorbox}
The implication problem for traditional INDs is **PSPACE-complete**.

**Good news**: the complexity does not hike up in the absence of attributes with a finite domain.

**Theorem**

*In the absence of attributes with a finite domain, the implication problem for CINDs is **PSPACE-complete**.*

In the general setting, however,

**Theorem**

*The implication problem for CINDs is **EXPTIME-complete** in the general setting.*
Finite axiomatizability of CINDs

There is a sound and complete inference system for traditional INDs (reflexivity, projection/permutation, transitivity),

**Theorem**

There is a *sound and complete* inference system for CINDs.

The inference system is more involved than its traditional counterpart.
Static Analysis: CINDs vs. INDs

In the absence of attributes with a finite domain:

<table>
<thead>
<tr>
<th></th>
<th>satisfiability</th>
<th>implication</th>
<th>finite axiomatizability</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIND</td>
<td>$O(1)$</td>
<td>PSPACE-complete</td>
<td>yes</td>
</tr>
<tr>
<td>IND</td>
<td>$O(1)$</td>
<td>PSPACE-complete</td>
<td>yes</td>
</tr>
</tbody>
</table>

General setting:

<table>
<thead>
<tr>
<th></th>
<th>satisfiability</th>
<th>implication</th>
<th>finite axiomatizability</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIND</td>
<td>$O(1)$</td>
<td>EXPTIME-complete</td>
<td>yes</td>
</tr>
<tr>
<td>IND</td>
<td>$O(1)$</td>
<td>PSPACE-complete</td>
<td>yes</td>
</tr>
</tbody>
</table>

The interaction between domain constraints and CINDs.
CFDs and CINDs taken together

We need both CFDs and CINDs for

- data cleaning
- schema mapping

For traditional FDs and INDs taken together,

- the satisfiability problem is in $O(1)$ time, and
- the implication problem is undecidable.
CFDs and CINDs taken together

We need both CFDs and CINDs for

- data cleaning
- schema mapping

For traditional FDs and INDs taken together,

- the satisfiability problem is in $O(1)$ time, and
- the implication problem is undecidable.

In contrast,

**Theorem**

For CFDs and CINDs taken together,

- the satisfiability problem becomes undecidable, and
- the implication problem remains undecidable.

The need for effective heuristic algorithms
Dependency propagation: The need

In data exchange or data integration, dependencies that hold on sources may only hold conditionally on the target data.

- **Sources:** two relations for customers in the UK and USA
  \[ R_S(\text{AC}: \text{int}, \text{phn}: \text{int}, \text{name}: \text{string}, \text{street}: \text{string}, \text{city}: \text{string}, \text{zip}: \text{string}) \]

- **A traditional FD on** \( R_{UK} \): \( \text{zip} \rightarrow \text{street} \)

- **View definition:** \( (R_{UK} \times (\text{CC}: 44)) \cup (R_{USA} \times (\text{CC}: 01)) \)

- **The FD no longer** holds on the target data
Dependency propagation: The need

In data exchange or data integration, dependencies that hold on sources may only hold conditionally on the target data.

- Sources: two relations for customers in the UK and USA
  \[ R_S(\text{AC}: \text{int}, \text{phn}: \text{int}, \text{name}: \text{string}, \text{street}: \text{string}, \text{city}: \text{string}, \text{zip}: \text{string}) \]

- A traditional FD on \( R_{UK} \): \( \text{zip} \rightarrow \text{street} \)

- View definition: \( (R_{UK} \times (\text{CC}: 44)) \cup (R_{USA} \times (\text{CC}: 01)) \)

- The FD no longer holds on the target data

- The FD is indeed propagated to the target, but as a CFD

\[
([\text{CC}, \text{zip}] \rightarrow [\text{street}], T_p)
\]

\[
\begin{array}{ccc}
\text{CC} & \text{zip} & \text{street} \\
44 & - & - \\
\end{array}
\]
Dependency propagation

- **Input:**
  - A set $\Sigma$ of source dependencies: FDs (or CFDs) on the sources
  - View definition $\sigma$: a query in (a fragment of) relational algebra
  - A view dependency $\varphi$

- **Question:** Is $\varphi$ propagated from $\Sigma$ via $\sigma$?

  For any source database $D$ that satisfies $\Sigma$, the view $\sigma(D)$ is guaranteed to satisfy $\varphi$.

Studied for propagation from source FDs to view FDs:


It is believed that the dependency propagation problem is

- in \textbf{PTIME} for views defined in terms of \textbf{SPCU} queries (selection, projection, Cartesian product, union),
- \textbf{undecidable} for views defined in relational algebra.
It is believed that the dependency propagation problem is

- in \text{PTIME} for views defined in terms of \text{SPCU} queries
  (selection, projection, Cartesian product, union),
- \text{undecidable} for views defined in relational algebra.

The \text{PTIME} result holds, but only in the absence of attributes with a finite domain:

\textbf{Theorem}

\textit{The propagation problem from source FDs to view FDs is already coNP-complete for SC views in the general setting.}

There is interaction between domain constraints and dependency propagation analysis.
Reasoning about matching dependencies (MDs)

Matching dependencies: if $C$ holds then identify $x$ and $y$.

Generic reasoning:

- A set $\Sigma$ of MDs entails another MD $\phi$, denoted by $\Sigma \models_m \phi$, if for any instance $D$ that satisfies $\Sigma$, $D$ satisfies $\phi$,
  - for all similarity and match relations satisfying their generic axioms (reflexivity, symmetric and subsuming equality for $\approx$; and additionally, transitivity and pairwise match for $\equiv$)

- The implication problem for MDs: to determine, given any $\Sigma$ and $\phi$, whether or not $\Sigma \models_m \phi$.

Logical consequence: no matter how matching rules are interpreted, if $\Sigma$ is enforced, then so must be $\phi$. 
Derived MDs can add value

- A set $\Sigma$ of given MDs:
  
  $\text{card}[LN] \rightleftharpoons \text{billing}[SN] \land \text{card}[addr] \rightleftharpoons \text{billing}[post] \land$
  
  $\text{card}[FN] \rightleftharpoons \text{billing}[FN] \rightarrow \text{card}[Y_c] \rightleftharpoons \text{billing}[Y_b]$

  $\text{card}[LN] \rightleftharpoons \text{billing}[SN] \land \text{card}[addr] \rightleftharpoons \text{billing}[post] \land$
  
  $\text{card}[FN] \approx_d \text{billing}[FN] \rightarrow \text{card}[Y_c] \rightleftharpoons \text{billing}[Y_b]$

  $\text{card}[tel] = \text{billing}[phn] \rightarrow \text{card}[addr] \rightleftharpoons \text{billing}[post]$

  $\text{card}[email] = \text{billing}[email] \rightarrow \text{card}[FN, LN] \rightleftharpoons \text{billing}[FN, SN]$

- Derived MDs: $\Sigma \models_m \phi$
  
  $\text{card}[email, addr] = \text{billing}[email, post] \rightarrow \text{card}[Y_c] \rightleftharpoons \text{billing}[Y_b]$

  $\text{card}[LN, tel] = \text{billing}[SN, phn] \land \text{card}[FN] \approx_d \text{billing}[FN]$
  
  $\rightarrow \text{card}[Y_c] \rightleftharpoons \text{billing}[Y_b]$
Derived MDs can add value

- A set $\Sigma$ of given MDs:

$$\text{card}[\text{LN}] \iff \text{billing}[\text{SN}] \land \text{card}[\text{addr}] \iff \text{billing}[\text{post}] \land \text{card}[\text{FN}] \iff \text{billing}[\text{FN}] \rightarrow \text{card}[Y_c] \iff \text{billing}[Y_b]$$

$$\text{card}[\text{LN}] \iff \text{billing}[\text{SN}] \land \text{card}[\text{addr}] \iff \text{billing}[\text{post}] \land \text{card}[\text{FN}] \approx_d \text{billing}[\text{FN}] \rightarrow \text{card}[Y_c] \iff \text{billing}[Y_b]$$

$$\text{card}[\text{tel}] = \text{billing}[\text{phn}] \rightarrow \text{card}[\text{addr}] \iff \text{billing}[\text{post}]$$

$$\text{card}[\text{email}] = \text{billing}[\text{email}] \rightarrow \text{card}[\text{FN, LN}] \iff \text{billing}[\text{FN, SN}]$$

- Derived MDs: $\Sigma \models_m \phi$

$$\text{card}[\text{email, addr}] = \text{billing}[\text{email, post}] \rightarrow \text{card}[Y_c] \iff \text{billing}[Y_b]$$

$$\text{card}[\text{LN, tel}] = \text{billing}[\text{SN, phn}] \land \text{card}[\text{FN}] \approx_d \text{billing}[\text{FN}]$$

$$\rightarrow \text{card}[Y_c] \iff \text{billing}[Y_b]$$

- When tuples differ in each of (LN, SN) and (addr, post), they can be identified via derived MDs, but not by the given MDs.
The implication problem for matching dependencies

**Derived MDs:**

\[
\begin{align*}
\text{card}[\text{email, addr}] &= \text{billing}[\text{email, post}] \rightarrow \text{card}[Y_c] \iff \text{billing}[Y_b] \\
\text{card}[\text{LN, tel}] &= \text{billing}[\text{SN, phn}] \land \text{card}[\text{FN}] \approx_d \text{billing}[\text{FN}] \\
&\rightarrow \text{card}[Y_c] \iff \text{billing}[Y_b]
\end{align*}
\]

- These derived MDs allow us to identify tuples based **solely** on the similarity metrics given on the source data.
- The implication analysis of MDs aims to derive matching rules on unreliable data.
The implication problem for matching dependencies

Derived MDs:

\[
\begin{align*}
card[\text{email, addr}] &= billing[\text{email, post}] \rightarrow card[Y_c] \Leftarrow billing[Y_b] \\
card[\text{LN, tel}] &= billing[\text{SN, phn}] \land card[\text{FN}] \approx_d billing[\text{FN}] \\
&\rightarrow card[Y_c] \Leftarrow billing[Y_b]
\end{align*}
\]

- These derived MDs allow us to identify tuples based solely on the similarity metrics given on the source data.
- The implication analysis of MDs aims to derive matching rules on unreliable data.

**Theorem**

The implication problem for matching dependencies is in \textit{PTIME}.
Conditional dependencies for capturing data inconsistencies
  ▶ Conditional functional dependencies (CFDs)
  ▶ Conditional inclusion dependencies (CINDs)
  ▶ Other extensions

Matching dependencies for object identification
  ▶ Object identification and matching rules
  ▶ Matching dependencies

Static analyses: New challenges
  ▶ Reasoning about conditional dependencies: Satisfiability, implication, axiomatizability, dependency propagation
  ▶ Inferring matching rules

Improving data quality with dependencies
  ▶ Data repairing (Arenas, Bertossi, Chomicki)
  ▶ Consistent querying answering (Arenas, Bertossi, Chomicki)
  ▶ Condensed representations of all repairs

Open research issues

- **Input:** a relational database $D$ and a set $\Sigma$ of dependencies
- **Output:** a repair $D'$ of $D$ w.r.t. $\Sigma$: $D' \models \Sigma$ (consistent), and $D'$ minimally differs from the original database $D$.

Example repair models:

- **X-repair:** maximal $D' \subseteq D$, $D' \models \Sigma$ (tuple deletions)
- **S-repair:** minimal $(D \setminus D') \cup (D' \setminus D)$ and $D' \models \Sigma$ (tuple insertions and deletions)
- **U-repair:** $D' \models \Sigma$ and minimal cost($D', D$) (value modifications).
Data repairing


- **Input**: a relational database $D$ and a set $\Sigma$ of dependencies
- **Output**: a repair $D'$ of $D$ w.r.t. $\Sigma$: $D' \models \Sigma$ (consistent), and $D'$ minimally differs from the original database $D$.

Example repair models:

- **$U$-repair**: $D' \models \Sigma$ and minimal cost($D'$, $D$) (value modifications). A simple example:

$$cost(D', D) = \sum_{t \in D, t' \in D'} \sum_{A \in R} w(t, A) \cdot dis(t[A], t'[A])$$

- $t'$: the updated version of tuple $t$;
- $w(t, A)$: the accuracy of the attribute $A$;
- $dis(u, v)$: the distance between values.
The repair checking problem

Given $\Sigma$, $D$, and $D'$, whether $D'$ is a repair of $D$ w.r.t. $\Sigma$?

**Theorem**

The repair checking problem is

- in $\text{PTIME}$ for denial constraints ($S$-repairs) [2];
- $\text{coNP}$-hard for universal dependencies, and in $\text{coNP}$ for any FO sentences ($S$-repairs) [2];
- in $\text{PTIME}$ for FDs and acyclic INDs ($X$-repairs) [1].
- $\text{coNP}$-complete for one FD and one IND together ($X$-repairs) [1];
- $\text{NP}$-complete for a fixed set of either FDs or INDs ($U$-repairs);
- ...

Heuristic for finding a candidate repair

- **Repairing**: given a database $D$ and a set $\Sigma$ of dependencies, it is to find a candidate repair $D'$ of $D$ w.r.t. $\Sigma$

- **Incremental repairing**: given $\Sigma$, $D$, $D'$ and updates $\Delta D$ to the database $D$, it is to find updates $\Delta D'$ to the repair $D'$

- **Data imputation by US national statistical agencies**:
  


- **For denial constraints (local, numerical values)**
  

Heuristic for finding a candidate repair

- For traditional FDs and INDs taken together


- For CFDs


**Performance guarantee** (precision, recall) with a high confidence?
Heuristic for finding a candidate repair

- For traditional FDs and INDs taken together

- For CFDs

Performance guarantee (precision, recall) with a high confidence?

Master Data Management (MDM): (incremental) repairing based on available master (reference) data $D_r$

- combination of object identification and data repairing;
- schema mapping;
- ...

Consistent query answering


- **Input**: a database $D$, a set $\Sigma$ of dependencies, and a query $Q$
- **Output**: certain answers to $Q$ in $D$ w.r.t. $\Sigma$.
  Tuples that are in the answers to $Q$ in each repair of $D$ w.r.t. $\Sigma$.

**Invited talks and surveys:**

- ...
Complexity bounds for X repair

Theorem

The consistent query answering problem is

- in $\text{PTIME}$ for denial constraints and quantifier-free CQ [1];
- in $\text{PTIME}$ for primary keys and a restricted class $C_{\text{tree}}$ of CQ [2];
- $\text{coNP}$-complete for denial constraints, and is already $\text{coNP}$-hard for a single primary key, for a class of Boolean CQ;
- in $\text{PTIME}$ for INDs alone and CQ [3];
- $\Pi_2^p$-complete for FDs and INDs taken together, for CQ [3];
- ...

Complexity bounds for $S$ repair

**Theorem**

The consistent query answering problem is

- $\mathcal{C}(\sigma, \times, -)$: $\Pi^P_2$-complete for universal constraints;
- $\mathcal{C}(\sigma, \times, -, \cup)$: in $\text{PTIME}$ for denial constraints, and $\Pi^P_2$-complete for universal constraints;
- $\mathcal{C}(\sigma, \pi)$: in $\text{PTIME}$ for primary keys; $\text{coNP}$-complete for denial constraints, and $\Pi^P_2$-complete for universal constraints;
- $\mathcal{C}(\sigma, \pi, \times)$: $\text{coNP}$-complete for primary keys, and $\Pi^P_2$-complete for universal constraints;
- $\mathcal{C}(\sigma, \pi, \times, -, \cup)$: $\text{coNP}$-complete for primary keys, and $\Pi^P_2$-complete for universal constraints.


...
Representation systems for incomplete information

- **Representation of possible instances:** a table $T$ with variables
  $$\text{rep}(T) = \{\mu(T) \mid \mu \text{ is a valuation of variables in } T\}$$

- **Strong representation system** for a query language $L$: for each representation $T$ and each $q \in L$, there exists a computable $\bar{q}(T)$ (representing $\{q(D) \mid D \in \text{rep}(T)\}$) such that
  $$\text{rep}(\bar{q}(T)) = q(\text{rep}(T))$$ – the possible answers
  e.g., conditional tables

- **Weak representation system** for $L$: representing the certain answers, e.g., naive tables

**Surveys:**


- **Input**: a database $D$ and a satisfiable set $\Sigma$ of full dependencies
- **Output**: a nucleus $G$, a single tableau with variables
  - representing all $U$-repairs of $D$ w.r.t. $\Sigma$: for each CQ query $q$, $q(G)$ yields the consistent answers to $q$ in $D$ w.r.t. $\Sigma$;
  - $G$ is homomorphic to all $U$-repairs;
  - for any tableau that is homomorphic to all $U$-repairs, it is also homomorphic to $G$;
  - for a fixed set of dependencies, $|G|$ can be exponential in $|D|$.
Condensed representations of all repairs: Other approaches

- **Answer sets of disjunctive logic programs:** for FO queries and full dependencies
  
  

- **World-set decompositions:** finite sets of possible worlds, via the product of decomposed relations
  
  

- ...
Conditional dependencies for capturing data inconsistencies
  - Conditional functional dependencies (CFDs)
  - Conditional inclusion dependencies (CINDs)
  - Other extensions

Matching dependencies for object identification
  - Object identification and matching rules
  - Matching dependencies

Static analyses: New challenges
  - Reasoning about conditional dependencies: Satisfiability, implication, axiomatizability, dependency propagation
  - Inferring matching rules

Improving data quality with dependencies
  - Data repairing (Arenas, Bertossi, Chomicki)
  - Consistent querying answering (Arenas, Bertossi, Chomicki)
  - Condensed representations of all repairs

Open research issues
Dependencies revisited

- To capture data inconsistencies: adding conditions
- To identify objects: incorporating similarity
- Revision of static analyses: satisfiability, implication, axiomatizability, dependency propagation
- ...

...
Dependencies revisited

- To capture data inconsistencies: adding conditions
- To identify objects: incorporating similarity
- Revision of static analyses: satisfiability, implication, axiomatizability, dependency propagation
- ...

To find practical use of dependencies in data quality tools
- Develop appropriate constraint languages for improving data quality: revising equality-generating dependencies (EGDs) and tuple-generating dependencies (TGDs)
- Integrate data repairing and object identification: reasoning about conditional dependencies and matching dependencies taken together
- Repairing algorithms with performance guarantee
- Consistent querying answering for conditional dependencies
- ...
Interactions with other lines of research

- **Incomplete information:**
  - Representation systems for **all repairs**?
  - **Tuple completeness** via extensions of TGDs: missing **tuples** in addition to missing values.


- ...
Interactions with other lines of research

- **Incomplete information:**
  - Representation systems for all repairs?
  - **Tuple completeness** via extensions of TGDs: missing tuples in addition to missing values.


- ... 

- **Date exchange and data integration:**
  - Adding context to schema matching: CINDs, extending TGDs with conditions
    
    $\text{(order(title, price; type = 'book') } \subseteq \text{ book(title, price))}$


  - Coping with unreliable data sources


  - ...
Interactions with other lines of research

- **Probabilistic data management:**
  - **Consistent query answering**
    
  
  - **Dependencies** for probabilistic data: soft keys?
  
  - ...
Interactions with other lines of research

- **Probabilistic data management:**
  - Consistent query answering
    
  - Dependencies for probabilistic data: soft keys?
  - ...

- **Provenance.** P. Buneman: Curated Dabatases
  - Provenance via *dependency propagation* analysis
  - ...

Wenfei Fan – Dependencies Revisited for Improving Data Quality
Interactions with other lines of research

- **Probabilistic data management:**
  - **Consistent query answering**
  - **Dependencies** for probabilistic data: soft keys?
  - ...

- **Provenance.** P. Buneman: Curated Databases
  - **Provenance via dependency propagation analysis**
  - ...

- **XML data cleaning:**
  - **XML constraints for data cleaning:** more intriguing
  - **Repairing algorithms and consistent query answering**
  - ...

Interactions with other lines of research

- **Probabilistic data management:**
  - Consistent query answering
    
  - Dependencies for probabilistic data: soft keys?
  - ... 

- **Provenance.** P. Buneman: Curated Databases
  - Provenance via dependency propagation analysis
  - ... 

- **XML data cleaning:**
  - XML constraints for data cleaning: more intriguing
  - Repairing algorithms and consistent query answering
    
  - ... 

A rich source of questions