Database Consistency: Logic-Based Approaches

Jan Chomicki
University at Buffalo and Warsaw University

June 25-29, 2007
Plan of the course

1. Integrity constraints

2. Consistent query answers

3. XML
Outline of Part I

1. Basic notions

2. Implication of dependencies

3. Axiomatization

4. Applications
   - Database design
   - Data exchange
   - Semantic query optimization
Integrity constraints (dependencies)

Database instance $D$:
- a finite first-order structure
- the information about the world
Integrity constraints (dependencies)

**Database instance $D$:**
- a finite first-order structure
- the information about the world

**Integrity constraints $\Sigma$:**
- first-order logic formulas
- the properties of the world

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Inconsistent database:
<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>
Integrity constraints (dependencies)

**Database instance $D$:**
- a finite first-order **structure**
- the information about the world

**Integrity constraints $\Sigma$:**
- first-order logic **formulas**
- the properties of the world

Satisfaction of constraints: $D \models \Sigma$

Formula **satisfaction** in a first-order structure.
Integrity constraints (dependencies)

Database instance $D$:
- a finite first-order structure
- the information about the world

Integrity constraints $\Sigma$:
- first-order logic formulas
- the properties of the world

Satisfaction of constraints: $D \models \Sigma$
Formula satisfaction in a first-order structure.

Consistent database: $D \models \Sigma$

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name $\rightarrow$ City Salary
Integrity constraints (dependencies)

Database instance $D$:
- a finite first-order structure
- the information about the world

Integrity constraints $\Sigma$:
- first-order logic formulas
- the properties of the world

Satisfaction of constraints: $D \models \Sigma$

Formula satisfaction in a first-order structure.

Consistent database: $D \models \Sigma$

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name $\rightarrow$ City Salary

Inconsistent database: $D \not\models \Sigma$

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name $\rightarrow$ City Salary
The need for integrity constraints

Roles of integrity constraints

- capture the **semantics** of data:
  - legal values of attributes
  - object identity
  - relationships, associations

- reduce data **errors** ⇒ **data quality**
- help in database **design**
- help in query **formulation**

(usually) no effect on query **semantics** but ...

- query **evaluation** and **analysis** are affected:
  - indexes, access paths
  - query containment and equivalence
  - semantic query optimization (SQO)

Examples

- **key** functional dependency: “every employee has a single address and salary”
- **denial** constraint: “no employee can earn more than her manager”
- **foreign key** constraint: “every manager is an employee”
Constraint enforcement

Enforced by application programs:
- Constraint checks inserted into code
- Code duplication and increased application complexity
- Error-prone: different applications can make different assumptions
- Prevent system-level optimizations

Enforced by DBMS:
- Constraint checks performed by DBMS ("factored out")
- Violating updates rolled back
- Leads to application simplification and reduces errors
- Enables query optimizations
- But... integrity checks are expensive and inflexible

Not enforced:
- Data comes from multiple, independent sources
- Long transactions with inconsistent intermediate states
- Enforcement too expensive
Constraint enforcement

Enforced by application programs

- constraint checks inserted into code
- code duplication and increased application complexity
- error-prone: different applications can make different assumptions
- prevent system-level optimizations

Enforced by DBMS
- constraint checks performed by DBMS (“factored out”)
- violating updates rolled back
- leads to application simplification and reduces errors
- enables query optimizations
- but ... integrity checks are expensive and inflexible

Not enforced
- data comes from multiple, independent sources
- long transactions with inconsistent intermediate states
- enforcement too expensive
Constraint enforcement

Enforced by application programs
- constraint checks inserted into code
- code duplication and increased application complexity
- error-prone: different applications can make different assumptions
- prevent system-level optimizations

Enforced by DBMS
- constraint checks performed by DBMS ("factored out")
- violating updates rolled back
- leads to application simplification and reduces errors
- enables query optimizations
- but ... integrity checks are expensive and inflexible
Constraint enforcement

Enforced by application programs
- constraint checks inserted into code
- code duplication and increased application complexity
- error-prone: different applications can make different assumptions
- prevent system-level optimizations

Enforced by DBMS
- constraint checks performed by DBMS ("factored out")
- violating updates rolled back
- leads to application simplification and reduces errors
- enables query optimizations
- but ... integrity checks are expensive and inflexible

Not enforced
- data comes from multiple, independent sources
- long transactions with inconsistent intermediate states
- enforcement too expensive
Basic issues

Implication

Given a set of ICs $\Sigma$ and an IC $\sigma$, does $D | D = \Sigma$ imply $D | D = \sigma$ for every database $D$?

Axiomatization

Can the notion of implication be “axiomatized”?

Inconsistent databases

1. How to construct a consistent database on the basis of an inconsistent one?
2. How to obtain information unaffected by inconsistency?
### Implication

Given a set of ICs $\Sigma$ and an IC $\sigma$, does $D \models \Sigma$ imply $D \models \sigma$ for every database $D$?
Basic issues

**Implication**

Given a set of ICs $\Sigma$ and an IC $\sigma$, does $D \models \Sigma$ imply $D \models \sigma$ for every database $D$?

**Axiomatization**

Can the notion of implication be “axiomatized”?
Basic issues

Implication
Given a set of ICs $\Sigma$ and an IC $\sigma$, does $D \models \Sigma$ imply $D \models \sigma$ for every database $D$?

Axiomatization
Can the notion of implication be “axiomatized”?

Inconsistent databases
1. How to construct a consistent database on the basis of an inconsistent one?
2. How to obtain information unaffected by inconsistency?
ICs in logical form

Atomic formulas

- relational (database) atoms
  \[ P(x_1, \ldots, x_k) \]
- equality atoms
  \[ x_1 = x_2 \]

- no constants

General form

\[ \forall x_1, \ldots, x_k. A_1 \land \cdots \land A_n \Rightarrow \exists y_1, \ldots, y_l. B_1 \land \cdots \land B_m. \]

Subclasses

- full dependencies: no existential variables \((l = 0)\)
- tuple-generating dependencies (TGDs): no equality atoms
- equality-generating dependencies (EGDs): \(m = 1, B_1\) is an equality atom
- functional dependencies (FDs): typed binary unirelational EGDs
- join dependencies (JDs): TGDs with LHS a multiway join
- denial constraints: \(l = 0, m = 0\)
- inclusion dependencies (INDs): \(n = m = 1\), no equality atoms
ICs in logical form

<table>
<thead>
<tr>
<th>Atomic formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>• relational (database) atoms $P(x_1, \ldots, x_k)$</td>
</tr>
<tr>
<td>• equality atoms $x_1 = x_2$</td>
</tr>
<tr>
<td>• no constants</td>
</tr>
</tbody>
</table>
ICs in logical form

Atomic formulas
- relational (database) atoms $P(x_1, \ldots, x_k)$
- equality atoms $x_1 = x_2$
- no constants

General form
$$\forall x_1, \ldots, x_k. \ A_1 \land \cdots \land A_n \Rightarrow \exists y_1, \ldots, y_l. \ B_1 \land \cdots \land B_m.$$
ICs in logical form

Atomic formulas
- relational (database) atoms \( P(x_1, \ldots, x_k) \)
- equality atoms \( x_1 = x_2 \)
- no constants

General form
\[
\forall x_1, \ldots, x_k. \; A_1 \land \cdots \land A_n \Rightarrow \exists y_1, \ldots, y_l. \; B_1 \land \cdots \land B_m.
\]

Subclasses
- full dependencies: no existential variables \( (l = 0) \)
- tuple-generating dependencies (TGDs): no equality atoms
- equality-generating dependencies (EGDs): \( m = 1 \), \( B_1 \) is an equality atom
- functional dependencies (FDs): typed binary unirelational EGDs
- join dependencies (JDs): TGDs with LHS a multiway join
- denial constraints: \( l = 0 \), \( m = 0 \)
- inclusion dependencies (INDs): \( n = m = 1 \), no equality atoms
Examples

Database schema $NAM(\text{Name}, \text{Address}, \text{Manager}), \ NAS(\text{Name}, \text{Address}, \text{Salary}), \ NM(\text{Name}, \text{Manager})$. 
Examples

Database schema $NAM(Name, Address, Manager)$, $NAS(Name, Address, Salary)$, $NM(Name, Manager)$.

**Full TGD**

$\forall n, a, m, s. \ NAS(n, a, s) \land NM(n, m) \Rightarrow NAM(n, a, m)$
Examples

Database schema \( NAM(\text{Name}, \text{Address}, \text{Manager}) \), \( NAS(\text{Name}, \text{Address}, \text{Salary}) \), \( NM(\text{Name}, \text{Manager}) \).

**Full TGD**
\[
\forall n, a, m, s. \ NAS(n, a, s) \land NM(n, m) \Rightarrow NAM(n, a, m)
\]

**Non-full TGD**
\[
\forall n, a, m. \ NAM(n, a, m) \Rightarrow \exists s. \ NAS(n, a, s)
\]

**Inclusion dependency (IND)**
\[
NAM[\text{Name}, \text{Address}] \subseteq NAS[\text{Name}, \text{Address}]
\]
Examples

Database schema $NAM(Name, Address, Manager)$, $NAS(Name, Address, Salary)$, $NM(Name, Manager)$.

Full TGD

$\forall n, a, m, s. \ NAS(n, a, s) \land NM(n, m) \Rightarrow NAM(n, a, m)$

Non-full TGD

$\forall n, a, m. \ NAM(n, a, m) \Rightarrow \exists s. \ NAS(n, a, s)$

Inclusion dependency (IND)

$NAM[Name, Address] \subseteq NAS[Name, Address]$  

EGD

$\forall n, a, m, a', m'. \ NAM(n, a, m) \land NAM(n, a', m') \Rightarrow a = a'$

Functional dependency (FD)

$Name \rightarrow Address$
Implication: from linear-time to undecidable

Functional dependencies

1 view each attribute as a propositional variable
2 view each dependency $A_1 \ldots A_k \rightarrow B \in \Sigma$ as a Horn clause
3 if $\sigma = C_1 \land \ldots \land C_d \Rightarrow D$, then $\neg \sigma = C_1 \land \ldots \land C_d \land \neg D$ consists of Horn clauses
4 thus $\Sigma \cup \neg \sigma$ is a set of Horn clauses whose (un)satisfiability can be tested in linear time (Dowling, Gallier [DG84])

Theorem (Chandra, Vardi [CV85])
The implication problem for functional dependencies together with inclusion dependencies is undecidable.
Functional dependencies

1. view each attribute as a propositional variable
2. view each dependency $A_1 \ldots A_k \rightarrow B \in \Sigma$ as a Horn clause $A_1 \land \cdots \land A_k \Rightarrow B$
3. if $\sigma = C_1 \land \cdots \land C_d \Rightarrow D$, then $\neg \sigma = C_1 \land \cdots \land C_d \land \neg D$ consists of Horn clauses
4. thus $\Sigma \cup \neg \sigma$ is a set of Horn clauses whose (un)satisfiability can be tested in linear time (Dowling, Gallier [DG84])
Implication: from linear-time to undecidable

Functional dependencies

1. view each attribute as a propositional variable
2. view each dependency $A_1 \ldots A_k \rightarrow B \in \Sigma$ as a Horn clause $A_1 \land \cdots \land A_k \Rightarrow B$
3. if $\sigma = C_1 \land \cdots \land C_d \Rightarrow D$, then $\neg \sigma = C_1 \land \cdots \land C_d \land \neg D$ consists of Horn clauses
4. thus $\Sigma \cup \neg \sigma$ is a set of Horn clauses whose (un)satisfiability can be tested in linear time (Dowling, Gallier [DG84])

Theorem (Chandra, Vardi [CV85])

The implication problem for functional dependencies together with inclusion dependencies is undecidable.
Implication in logic

No restriction to **finite structures**.
Implication in logic

No restriction to finite structures.

Finite and unrestricted implication

- coincide for full dependencies
- if they coincide, then they are decidable
- but not vice versa (FDs and unary INDs)

Counterexample
\[ \Sigma = \{ A \rightarrow B, R[A] \subseteq R[B] \} \]
\[ \sigma = R[B] \subseteq R[A] \]
No restriction to finite structures.

Finite and unrestricted implication
- coincide for full dependencies
- if they coincide, then they are decidable
- but not vice versa (FDs and unary INDs)

Counterexample
\[\Sigma = \{A \rightarrow B, R[A] \subseteq R[B]\}\]
\[\sigma = R[B] \subseteq R[A]\]
No restriction to **finite structures**.

**Finite and unrestricted implication**

- coincide for full dependencies
- if they coincide, then they are decidable
- but not vice versa (FDs and *unary* INDs)

**Counterexample**

\[ \Sigma = \{ A \rightarrow B, R[A] \subseteq R[B] \} \]
\[ \sigma = R[B] \subseteq R[A] \]
Implication in logic

No restriction to finite structures.

Finite and unrestricted implication

- coincide for full dependencies
- if they coincide, then they are decidable
- but not vice versa (FDs and unary INDs)

Counterexample

\[ \Sigma = \{ A \rightarrow B, R[A] \subseteq R[B] \} \]
\[ \sigma = R[B] \subseteq R[A] \]

Finite and unrestricted implication do not have to coincide.
Deciding the implication of full dependencies using chase

1. apply chase steps using the dependencies in $\Sigma$ nondeterministically, obtaining a sequence of dependencies
2. $\tau_0 = \sigma, \tau_1, \ldots, \tau_n$
3. stop when no chase steps can be applied to $\tau_n$ (a terminal chase sequence)
4. if $\tau_n$ is trivial, then $\Sigma$ implies $\sigma$
5. otherwise, $\Sigma$ does not imply $\sigma$

Trivial dependencies
- tgd: LHS contains RHS
- egd: RHS $\equiv x = x$

Fundamental properties of the chase
- Terminal chase sequence $\tau_0 = \sigma, \tau_1, \ldots, \tau_n$: the LHS of $\tau_n$, viewed as a database $D_n$, satisfies $\Sigma$ if $\tau_n$ is nontrivial, then $D_n$ violates $\sigma$
- the order of chase steps does not matter
Deciding the implication of full dependencies using chase

1. **apply** chase steps using the dependencies in $\Sigma$ nondeterministically, obtaining a sequence of dependencies $\tau_0 = \sigma, \tau_1, \ldots, \tau_n$

2. stop when no chase steps can be applied to $\tau_n$ (a terminal chase sequence)

3. if $\tau_n$ is **trivial**, then $\Sigma$ implies $\sigma$

4. otherwise, $\Sigma$ does not imply $\sigma$
Deciding the implication of full dependencies using chase

1. **apply** chase steps using the dependencies in $\Sigma$ nondeterministically, obtaining a sequence of dependencies $\tau_0 = \sigma, \tau_1, \ldots, \tau_n$
2. stop when no chase steps can be applied to $\tau_n$ (a terminal chase sequence)
3. if $\tau_n$ is trivial, then $\Sigma$ implies $\sigma$
4. otherwise, $\Sigma$ does not imply $\sigma$

**Trivial dependencies**

- **tgd**: LHS contains RHS
- **egd**: RHS $\equiv x = x$
Deciding the implication of full dependencies using chase

1. apply chase steps using the dependencies in $\Sigma$ nondeterministically, obtaining a sequence of dependencies $\tau_0 = \sigma, \tau_1, \ldots, \tau_n$
2. stop when no chase steps can be applied to $\tau_n$ (a terminal chase sequence)
3. if $\tau_n$ is trivial, then $\Sigma$ implies $\sigma$
4. otherwise, $\Sigma$ does not imply $\sigma$

Trivial dependencies

- tgd: LHS contains RHS
- egd: RHS $\equiv x = x$

Fundamental properties of the chase

Terminal chase sequence $\tau_0 = \sigma, \tau_1, \ldots, \tau_n$:

- the LHS of $\tau_n$, viewed as a database $D_n$, satisfies $\Sigma$
- if $\tau_n$ is nontrivial, then $D_n$ violates $\sigma$
- the order of chase steps does not matter
A chase sequence $\tau_0 = \sigma, \tau_1, \ldots$
A chase sequence $\tau_0 = \sigma, \tau_1, \ldots$.

**Applying a chase step using a tgd $C$**

1. view the LHS of $\tau_j$ as a database $D_j$
2. find a substitution $h$ that (1) $h$ makes the LHS of $C$ true in $D_j$, and (2) $h$ cannot be extended to a substitution that makes the RHS of $C$ true in that instance
3. apply $h$ to the RHS of $C$
4. add the resulting facts to the LHS of $\tau_j$, obtaining $\tau_{j+1}$
Chase steps

A chase sequence \( \tau_0 = \sigma, \tau_1, \ldots \).

### Applying a chase step using a tgd \( C \)
1. view the LHS of \( \tau_j \) as a database \( D_j \)
2. find a substitution \( h \) that (1) \( h \) makes the LHS of \( C \) true in \( D_j \), and (2) \( h \) cannot be extended to a substitution that makes the RHS of \( C \) true in that instance
3. apply \( h \) to the RHS of \( C \)
4. add the resulting facts to the LHS of \( \tau_j \), obtaining \( \tau_{j+1} \)

### Applying a chase step using an egd \( C \)
1. view the LHS of \( \tau_j \) as a database \( D_j \)
2. RHS of \( C \equiv x_1 = x_2 \)
3. find a substitution \( h \) such that makes the LHS of \( C \) true in \( D_j \) and \( h(x_1) \neq h(x_2) \)
4. replace all the occurrences of \( h(x_2) \) in \( \tau_j \) by \( h(x_1) \), obtaining \( \tau_{j+1} \)
Chase in action

Integrity constraints

\[ C_1 = \forall x, y. P(x, y) \Rightarrow R(x, y) \]

\[ C_2 = \forall x, y, z. R(x, y) \land R(x, z) \Rightarrow y = z \]

\[ C_3 = \forall x, y, z. P(x, y) \land P(x, z) \Rightarrow y = z \]

Goal

Show that \( \{ C_1, C_2 \} \) implies \( C_3 \).

Terminal chase sequence

\[ \tau_0 = \{ P(x, y) \land P(x, z) \Rightarrow y = z \} \]

\[ \tau_1 = \{ P(x, y) \land P(x, z) \land R(x, y) \Rightarrow y = z \} \]

\[ \tau_2 = \{ P(x, y) \land P(x, z) \land R(x, y) \land R(x, z) \Rightarrow y = z \} \]

\[ \tau_3 = \{ P(x, y) \land R(x, y) \Rightarrow y = y \} \]: a trivial dependency
Chase in action

**Integrity constraints**

\[ C_1 = \forall x, y. \ P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. \ R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. \ P(x, y) \land P(x, z) \Rightarrow y = z \]
Chase in action

Integrity constraints

\[ C_1 = \forall x, y. \ P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. \ R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. \ P(x, y) \land P(x, z) \Rightarrow y = z \]

Goal

Show that \( \{ C_1, C_2 \} \) implies \( C_3 \).
Chase in action

Integrity constraints

\[ C_1 = \forall x, y. P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. P(x, y) \land P(x, z) \Rightarrow y = z \]

Goal

Show that \( \{ C_1, C_2 \} \) implies \( C_3 \).

Terminal chase sequence
Chase in action

**Integrity constraints**

\[ C_1 = \forall x, y. \ P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. \ R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. \ P(x, y) \land P(x, z) \Rightarrow y = z \]

**Goal**

Show that \( \{ C_1, C_2 \} \) implies \( C_3 \).

**Terminal chase sequence**

\[ \tau_0 = \{ P(x, y) \land P(x, z) \Rightarrow y = z \} \]
Chase in action

Integrity constraints

\[ C_1 = \forall x, y. \ P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. \ R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. \ P(x, y) \land P(x, z) \Rightarrow y = z \]

Goal

Show that \( \{ C_1, C_2 \} \) implies \( C_3 \).

Terminal chase sequence

\[ \tau_0 = \{ P(x, y) \land P(x, z) \Rightarrow y = z \} \]
\[ \tau_1 = \{ P(x, y) \land P(x, z) \land R(x, y) \Rightarrow y = z \} \]
Chase in action

Integrity constraints

\[ C_1 = \forall x, y. \ P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. \ R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. \ P(x, y) \land P(x, z) \Rightarrow y = z \]

Goal

Show that \( \{ C_1, C_2 \} \) implies \( C_3 \).

Terminal chase sequence

\[ \tau_0 = \{ P(x, y) \land P(x, z) \Rightarrow y = z \} \]
\[ \tau_1 = \{ P(x, y) \land P(x, z) \land R(x, y) \Rightarrow y = z \} \]
\[ \tau_2 = \{ P(x, y) \land P(x, z) \land R(x, y) \land R(x, z) \Rightarrow y = z \} \]
Chase in action

### Integrity constraints

\[ C_1 = \forall x, y. \ P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. \ R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. \ P(x, y) \land P(x, z) \Rightarrow y = z \]

### Goal

Show that \( \{ C_1, C_2 \} \) implies \( C_3 \).

### Terminal chase sequence

\[ \tau_0 = \{ P(x, y) \land P(x, z) \Rightarrow y = z \} \]
\[ \tau_1 = \{ P(x, y) \land P(x, z) \land R(x, y) \Rightarrow y = z \} \]
\[ \tau_2 = \{ P(x, y) \land P(x, z) \land R(x, y) \land R(x, z) \Rightarrow y = z \} \]
\[ \tau_3 = \{ P(x, y) \land R(x, y) \Rightarrow y = y \} \]
Chase in action

Integrity constraints

\[ C_1 = \forall x, y. P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. P(x, y) \land P(x, z) \Rightarrow y = z \]

Goal

Show that \{ C_1, C_2 \} implies \( C_3 \).

Terminal chase sequence

\[ \tau_0 = \{ P(x, y) \land P(x, z) \Rightarrow y = z \} \]
\[ \tau_1 = \{ P(x, y) \land P(x, z) \land R(x, y) \Rightarrow y = z \} \]
\[ \tau_2 = \{ P(x, y) \land P(x, z) \land R(x, y) \land R(x, z) \Rightarrow y = z \} \]
\[ \tau_3 = \{ P(x, y) \land R(x, y) \Rightarrow y = y \}: \text{ a trivial dependency} \]
A general perspective

Computational complexity

Testing implication of full dependencies is:

in EXPTIME (using chase)

EXPTIME-complete (Chandra et al. [CLM81])

First-order logic

implication of

\( \sigma \) by \( \Sigma = \{\sigma_1, \ldots, \sigma_k\} \)

is equivalent to the unsatisfiability of the

formula \( \Phi_{\Sigma, \sigma} \equiv \sigma_1 \land \cdots \land \sigma_k \land \neg \sigma \)

for full dependencies, the formulas \( \Phi_{\Sigma, \sigma} \) are of the form

\[ \exists^* \forall^* \varphi \]

where \( \varphi \) is

quantifier-free (Bernays-Sch"{o}nkel class)

Bernays-Sch"{o}nkel formulas have the finite-model property and their satisfiability is

in NEXPTIME

Theorem proving

Chase corresponds to a combination of hyperresolution and paramodulation.
A general perspective

Computational complexity

Testing implication of full dependencies is:
- in EXPTIME (using chase)
- EXPTIME-complete (Chandra et al. [CLM81])
A general perspective

Computational complexity

Testing implication of full dependencies is:

- in EXPTIME (using chase)
- EXPTIME-complete (Chandra et al. [CLM81])

First-order logic

- implication of \( \sigma \) by \( \Sigma = \{\sigma_1, \ldots, \sigma_k\} \) is equivalent to the unsatisfiability of the formula \( \Phi_{\Sigma,\sigma} \equiv \sigma_1 \land \cdots \land \sigma_k \land \neg \sigma \)
- for full dependencies, the formulas \( \Phi_{\Sigma,\sigma} \) are of the form \( \exists^* \forall^* \phi \) where \( \phi \) is quantifier-free (Bernays-Schöfinkel class)
- Bernays-Schöfinkel formulas have the finite-model property and their satisfiability is in NEXPTIME
A general perspective

Computational complexity

Testing implication of full dependencies is:
- in EXPTIME (using chase)
- EXPTIME-complete (Chandra et al. [CLM81])

First-order logic

- implication of \( \sigma \) by \( \Sigma = \{\sigma_1, \ldots, \sigma_k\} \) is equivalent to the unsatisfiability of the formula \( \Phi_{\Sigma,\sigma} \equiv \sigma_1 \land \cdots \land \sigma_k \land \neg \sigma \)
- for full dependencies, the formulas \( \Phi_{\Sigma,\sigma} \) are of the form \( \exists^* \forall^* \phi \) where \( \phi \) is quantifier-free (Bernays-Schöfinkel class)
- Bernays-Schöfinkel formulas have the finite-model property and their satisfiability is in NEXPTIME

Theorem proving

Chase corresponds to a combination of hyperresolution and paramodulation.
Axiomatization

Inference rules specific to classes of dependencies guarantee closure: only dependencies from the same class are derived.

Properties

Inference rules capture finite or unrestricted implication:

- **Soundness**: all the dependencies derived from a given set $\Sigma$ are implied by $\Sigma$.
- **Completeness**: all the dependencies implied by $\Sigma$ can be derived from $\Sigma$.

A finite set of rules $\Rightarrow$ implication is decidable (but not vice versa).
Inference rules

- specific to classes of dependencies
- guarantee **closure**: only dependencies from the same class are derived
- **bounded** number of premises
Inference rules

- specific to classes of dependencies
- guarantee closure: only dependencies from the same class are derived
- bounded number of premises

Properties

Inference rules capture finite or unrestricted implication:

- soundness: all the dependencies derived from a given set $\Sigma$ are implied by $\Sigma$
- completeness: all the dependencies implied by $\Sigma$ can be derived from $\Sigma$
- finite set of rules $\Rightarrow$ implication decidable (but not vice versa)
Axiomatizing INDs

1. Reflexivity: $R[X] \subseteq R[X]$

2. Projection and permutation: If $R[A_1,...,A_m] \subseteq S[B_1,...,B_m]$, then $R[A_{i_1},...,A_{i_k}] \subseteq S[B_{i_1},...,B_{i_k}]$ for every sequence $i_1,...,i_k$ of distinct integers in $\{1,...,m\}$.

3. Transitivity: If $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$, then $R[X] \subseteq T[Z]$.

A derivation

Schemas $R(ABC)$ and $S(AB)$:

1. $S[AB] \subseteq R[AB]$ (given IND)
2. $R[C] \subseteq S[A]$ (given IND)
3. $S[A] \subseteq R[A]$ (from (1))
4. $R[C] \subseteq R[A]$ (from (2) and (3))
Example axiomatization

### Axiomatizing INDs

1. **Reflexivity**: $R[X] \subseteq R[X]$

2. **Projection and permutation**: If $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$, then $R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}]$ for every sequence $i_1, \ldots, i_k$ of distinct integers in $\{1, \ldots, m\}$.

3. **Transitivity**: If $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$, then $R[X] \subseteq T[Z]$.
Example axiomatization

Axiomatizing INDs

1. Reflexivity: $R[X] \subseteq R[X]$

2. Projection and permutation: If $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$, then $R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}]$ for every sequence $i_1, \ldots, i_k$ of distinct integers in $\{1, \ldots, m\}$.

3. Transitivity: If $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$, then $R[X] \subseteq T[Z]$.

A derivation

Schemas $R(ABC)$ and $S(AB)$:
Axiomatizing INDs

1. Reflexivity: \( R[X] \subseteq R[X] \)

2. Projection and permutation: If \( R[A_1, \ldots A_m] \subseteq S[B_1, \ldots B_m] \), then \( R[A_{i_1}, \ldots A_{i_k}] \subseteq S[B_{i_1}, \ldots B_{i_k}] \) for every sequence \( i_1, \ldots, i_k \) of distinct integers in \( \{1, \ldots, m\} \).

3. Transitivity: If \( R[X] \subseteq S[Y] \) and \( S[Y] \subseteq T[Z] \), then \( R[X] \subseteq T[Z] \).

A derivation

Schemas \( R(ABC) \) and \( S(AB) \):

1. \( S[AB] \subseteq R[AB] \) \( (\text{given IND}) \)
### Axiomatizing INDs

1. **Reflexivity**: \( R[X] \subseteq R[X] \)

2. **Projection and permutation**: If \( R[A_1, \ldots A_m] \subseteq S[B_1, \ldots B_m] \), then
   \[ R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}] \]
   for every sequence \( i_1, \ldots, i_k \) of distinct integers in \( \{1, \ldots, m\} \).

3. **Transitivity**: If \( R[X] \subseteq S[Y] \) and \( S[Y] \subseteq T[Z] \), then \( R[X] \subseteq T[Z] \).

### A derivation

**Schemas** \( R(ABC) \) and \( S(AB) \):

1. \( S[AB] \subseteq R[AB] \) (given IND)
2. \( R[C] \subseteq S[A] \) (given IND)
Axiomatizing INDs

1. **Reflexivity:** \( R[X] \subseteq R[X] \)

2. **Projection and permutation:** If \( R[A_1, \ldots A_m] \subseteq S[B_1, \ldots B_m] \), then \( R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}] \) for every sequence \( i_1, \ldots, i_k \) of distinct integers in \( \{1, \ldots, m\} \).

3. **Transitivity:** If \( R[X] \subseteq S[Y] \) and \( S[Y] \subseteq T[Z] \), then \( R[X] \subseteq T[Z] \).

A derivation

Schemas \( R(ABC) \) and \( S(AB) \):

1. \( S[AB] \subseteq R[AB] \) (given IND)
2. \( R[C] \subseteq S[A] \) (given IND)
3. \( S[A] \subseteq R[A] \) (from (1))
Example axiomatization

Axiomatizing INDs

1. Reflexivity: $R[X] \subseteq R[X]$

2. Projection and permutation: If $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$, then $R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}]$ for every sequence $i_1, \ldots, i_k$ of distinct integers in $\{1, \ldots, m\}$.

3. Transitivity: If $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$, then $R[X] \subseteq T[Z]$.

A derivation

Schemas $R(ABC)$ and $S(AB)$:

(1) $S[AB] \subseteq R[AB]$ (given IND)

(2) $R[C] \subseteq S[A]$ (given IND)

(3) $S[A] \subseteq R[A]$ (from (1))

(4) $R[C] \subseteq R[A]$ (from (2) and (3))
### Review of results

<table>
<thead>
<tr>
<th></th>
<th>Implication</th>
<th>Axiomatization</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDs</td>
<td>PTIME</td>
<td>Finite</td>
</tr>
<tr>
<td>INDs</td>
<td>PSPACE-complete</td>
<td>Finite</td>
</tr>
<tr>
<td>FDs + INDs</td>
<td>Undecidable</td>
<td>No</td>
</tr>
<tr>
<td>Full (typed) dependencies</td>
<td>EXPTIME-complete</td>
<td>Yes</td>
</tr>
<tr>
<td>Join dependencies</td>
<td>NP-complete</td>
<td>No</td>
</tr>
<tr>
<td>First-order logic</td>
<td>Undecidable</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Keys

A set of attributes $X \subseteq U$ is a key with respect to a set of FDs $\Sigma$ if:

$\Sigma$ implies $X \rightarrow U$ for no proper subset $Y$ of $X$, $\Sigma$ implies $Y \rightarrow U$.

Decomposition

A decomposition $R = (R_1, \ldots, R_n)$ of a schema $R$ has the lossless join property with respect to a set of FDs $\Sigma$ iff $\Sigma$ implies the join dependency $\Delta \Join [R]$.

Decomposition $(R_1, R_2)$ of $R(ABC)$

Relation schemas:

$R_1(AB)$ with FD $A \rightarrow B$,

$R_2(AC)$.

Terminal chase sequence:

$R(x,y,z') \land R(x,y',z) \Rightarrow R(x,y,z)$

given JD $R(x,y,z') \land R(x,y,z) \Rightarrow R(x,y,z)$

chase with $A \rightarrow B$.
A set of attributes $X \subseteq U$ is a key with respect to a set of FDs $\Sigma$ if:

- $\Sigma$ implies $X \rightarrow U$
- for no proper subset $Y$ of $X$, $\Sigma$ implies $Y \rightarrow U$
Keys

A set of attributes $X \subseteq U$ is a key with respect to a set of FDs $\Sigma$ if:

- $\Sigma$ implies $X \rightarrow U$
- for no proper subset $Y$ of $X$, $\Sigma$ implies $Y \rightarrow U$

Decomposition

A decomposition $R = (R_1, \ldots, R_n)$ of a schema $R$ has the lossless join property with respect to a set of FDs $\Sigma$ iff $\Sigma$ implies the join dependency $\Join [R]$. 
Application: database design

**Keys**

A set of attributes $X \subseteq U$ is a key with respect to a set of FDs $\Sigma$ if:

- $\Sigma$ implies $X \rightarrow U$
- for no proper subset $Y$ of $X$, $\Sigma$ implies $Y \rightarrow U$

**Decomposition**

A decomposition $\mathcal{R} = (R_1, \ldots, R_n)$ of a schema $R$ has the lossless join property with respect to a set of FDs $\Sigma$ iff $\Sigma$ implies the join dependency $\Join [\mathcal{R}]$.

**Decomposition $(R_1, R_2)$ of $R(ABC)$**

Relation schemas: $R_1(AB)$ with FD $A \rightarrow B$, $R_2(AC)$. 
Keys

A set of attributes $X \subseteq U$ is a key with respect to a set of FDs $\Sigma$ if:

- $\Sigma$ implies $X \rightarrow U$
- for no proper subset $Y$ of $X$, $\Sigma$ implies $Y \rightarrow U$

Decomposition

A decomposition $\mathcal{R} = (R_1, \ldots, R_n)$ of a schema $R$ has the **lossless join property** with respect to a set of FDs $\Sigma$ iff $\Sigma$ implies the join dependency $\bowtie [\mathcal{R}]$.

Decomposition $(R_1, R_2)$ of $R(ABC)$

Relation schemas: $R_1(AB)$ with FD $A \rightarrow B$, $R_2(AC)$.
Terminal chase sequence:
Keys
A set of attributes $X \subseteq U$ is a key with respect to a set of FDs $\Sigma$ if:
- $\Sigma$ implies $X \rightarrow U$
- for no proper subset $Y$ of $X$, $\Sigma$ implies $Y \rightarrow U$

Decomposition
A decomposition $\mathcal{R} = (R_1, \ldots, R_n)$ of a schema $R$ has the lossless join property with respect to a set of FDs $\Sigma$ iff $\Sigma$ implies the join dependency $\bowtie [\mathcal{R}]$.

Decomposition $(R_1, R_2)$ of $R(ABC)$
Relation schemas: $R_1(AB)$ with FD $A \rightarrow B$, $R_2(AC)$.
Terminal chase sequence:
$$R(x, y, z') \land R(x, y', z) \Rightarrow R(x, y, z)$$ given JD
Application: database design

Keys

A set of attributes \( X \subseteq U \) is a key with respect to a set of FDs \( \Sigma \) if:
- \( \Sigma \) implies \( X \rightarrow U \)
- for no proper subset \( Y \) of \( X \), \( \Sigma \) implies \( Y \rightarrow U \)

Decomposition

A decomposition \( \mathcal{R} = (R_1, \ldots, R_n) \) of a schema \( R \) has the lossless join property with respect to a set of FDs \( \Sigma \) iff \( \Sigma \) implies the join dependency \( \Join \mathcal{R} \).

Decomposition \( (R_1, R_2) \) of \( R(ABC) \)

Relation schemas: \( R_1(AB) \) with FD \( A \rightarrow B \), \( R_2(AC) \).

Terminal chase sequence:

\[
R(x, y, z') \land R(x, y', z) \Rightarrow R(x, y, z) \quad \text{given JD}
\]

\[
R(x, y, z') \land R(x, y, z) \Rightarrow R(x, y, z) \quad \text{chase with } A \rightarrow B
\]
Goal

Exchange of data between independent databases with different schemas.
Application: data exchange

Goal
Exchange of data between independent databases with different schemas.

Setting for data exchange
- source and target schemas
- source-to-target dependencies: describe how the data is mapped between source and target
- target integrity constraints
Application: data exchange

Goal
Exchange of data between independent databases with different schemas.

Setting for data exchange
- source and target schemas
- source-to-target dependencies: describe how the data is mapped between source and target
- target integrity constraints

Data exchange is a specific scenario for data integration, in which a target instance is constructed.
Constraints and solutions

\( \phi_S, \phi_T, \psi_T \) are conjunctions of relation atomic formulas over source and target.

### Source-to-target dependencies \( \Sigma_{st} \)
- tuple-generating dependencies: \( \forall x (\phi_S(x) \Rightarrow \exists y \psi_T(x, y)) \).

### Target integrity constraints \( \Sigma_t \)
- tuple-generating dependencies (tgds): \( \forall x (\phi_T(x) \Rightarrow \exists y \psi_T(x, y)) \)
- equality-generating dependencies: \( \forall x (\phi_T(x) \Rightarrow x_1 = x_2) \).
Constraints and solutions

$\phi_S$, $\phi_T$, $\psi_T$ are conjunctions of relation atomic formulas over source and target.

### Source-to-target dependencies $\Sigma_{st}$
- **tuple-generating dependencies**: $\forall x (\phi_S(x) \Rightarrow \exists y \psi_T(x, y))$.

### Target integrity constraints $\Sigma_t$
- **tuple-generating dependencies (tgds)**: $\forall x (\phi_T(x) \Rightarrow \exists y \psi_T(x, y))$
- **equality-generating dependencies**: $\forall x (\phi_T(x) \Rightarrow x_1 = x_2)$.

### Solution
Given a source instance $I$, a target instance $J$ is
- a **solution** for $I$ if $J$ satisfies $\Sigma_t$ and $(I, J)$ satisfy $\Sigma_{st}$
- a **universal solution** for $I$ if it is a solution for $I$ and there is a homomorphism from it to any other solution for $I$
- solutions can contain **labelled nulls**
Constraints and solutions

\( \phi_S, \phi_T, \psi_T \) are conjunctions of relation atomic formulas over source and target.

**Source-to-target dependencies** \( \Sigma_{st} \)

- **tuple-generating dependencies**: \( \forall x (\phi_S(x) \Rightarrow \exists y \psi_T(x, y)) \).

**Target integrity constraints** \( \Sigma_t \)

- **tuple-generating dependencies (tgds)**: \( \forall x (\phi_T(x) \Rightarrow \exists y \psi_T(x, y)) \)
- **equality-generating dependencies**: \( \forall x (\phi_T(x) \Rightarrow x_1 = x_2) \).

**Solution**

Given a source instance \( I \), a target instance \( J \) is

- a **solution** for \( I \) if \( J \) satisfies \( \Sigma_t \) and \( (I, J) \) satisfy \( \Sigma_{st} \)
- a **universal solution** for \( I \) if it is a solution for \( I \) and there is a homomorphism from it to any other solution for \( I \)
- solutions can contain **labelled nulls**

There may be multiple solutions.
Query evaluation (Fagin et al.[FKMP05])

**Certain answer**

Given a query $Q$ and a source instance $I$, a tuple $t$ is a **certain answer** with respect to $I$ if $t$ is an answer to $Q$ in every solution $J$ for $I$. 
Certainly answer

Given a query \( Q \) and a source instance \( I \), a tuple \( t \) is a certain answer with respect to \( I \) if \( t \) is an answer to \( Q \) in every solution \( J \) for \( I \).

Conjunctive queries

- relational calculus: \( \exists, \wedge \)
- relational algebra: \( \sigma, \pi, \times \)
**Certain answer**

Given a query $Q$ and a source instance $I$, a tuple $t$ is a certain answer with respect to $I$ if $t$ is an answer to $Q$ in every solution $J$ for $I$.

**Conjunctive queries**

- relational calculus: $\exists, \land$
- relational algebra: $\sigma, \pi, \times$

**Query evaluation**

1. construct any universal solution $J_0$
2. evaluate the query over $J_0$
3. discard answers with nulls
4. the above returns certain answers for unions of conjunctive queries without inequalities
Apply a variant of the chase [AHV95] to the source instance using target and source-to-target dependencies, obtaining a sequence of instances $I_0 = I, I_1, \ldots, I_n, \ldots$. 
Apply a variant of the chase [AHV95] to the source instance using target and source-to-target dependencies, obtaining a sequence of instances \( I_0 = I, I_1, \ldots, I_n, \ldots \).

### Chasing a tgd \( C \)

1. find a substitution \( h \) that (1) \( h \) makes the LHS of \( C \) true in the constructed instance \( I_j \), and (2) \( h \) cannot be extended to a substitution that makes the RHS of \( C \) true in that instance
2. apply \( h \) to the RHS of \( C \), mapping the existentially quantified variables to fresh labelled nulls
3. add the resulting facts to \( I_j \), obtaining \( I_{j+1} \).
Building a universal solution [FKMP05]

Apply a variant of the chase [AHV95] to the source instance using target and
source-to-target dependencies, obtaining a sequence of instances \( I_0 = I, I_1, \ldots, I_n, \ldots \).

**Chasing a tgd \( C \)**

1. find a substitution \( h \) that (1) \( h \) makes the LHS of \( C \) true in the constructed instance \( I_j \), and (2) \( h \) cannot be extended to a substitution that makes the RHS of \( C \) true in that instance
2. apply \( h \) to the RHS of \( C \), mapping the existentially quantified variables to fresh labelled nulls
3. add the resulting facts to \( I_j \), obtaining \( I_{j+1} \).

**Chasing an egd \( C \)**

Find a substitution \( h \) such that makes the LHS of \( C \) true in \( I_j \) and \( h(x_1) \neq h(x_2) \):
- if \( h(x_1) \) and \( h(x_2) \) are constants, then FAILURE
- otherwise, identify \( h(x_1) \) and \( h(x_2) \) in \( I_j \) (preferring constants), obtaining \( I_{j+1} \).
Chase at work

Source and target databases

Source:
Emp (N, A), Num (N, Id)

Target:
Name (Id, N), Addr (Id, A)

Source-to-target dependencies
∀ n, a. Emp(n, a) ⇒ ∃ id. Name(id, n) ∧ Addr(id, a)

∀ n, a, id. Emp(n, a) ∧ Num(n, id) ⇒ Name(id, n)

Target constraints
Name: N → Id, Id → N, Addr: Id → A.

Chase sequence

I₀ = {Emp(Li, LA), Num(Li, 111)}
I¹ = {Emp(Li, LA), Num(Li, 111), Name(id₁, Li), Addr(id₁, LA)}
I² = {Emp(Li, LA), Num(Li, 111), Name(id₁, Li), Addr(id₁, LA), Name(111, Li)}
I³ = {Emp(Li, LA), Num(Li, 111), Name(111, Li), Addr(111, LA)}
Chase at work

Source and target databases

**Source:** \(\text{Emp}(N, A), \text{Num}(N, Id)\)  
**Target:** \(\text{Name}(Id, N), \text{Addr}(Id, A)\)
## Chase at work

### Source and target databases

**Source:** $\text{Emp}(N, A), \text{Num}(N, Id)$  
**Target:** $\text{Name}(Id, N), \text{Addr}(Id, A)$

### Source-to-target dependencies

\[
\forall n, a. \ \text{Emp}(n, a) \Rightarrow \exists id. \ \text{Name}(id, n) \land \text{Addr}(id, a)
\]

\[
\forall n, a, id. \ \text{Emp}(n, a) \land \text{Num}(n, id) \Rightarrow \text{Name}(id, n)
\]

### Target constraints

$\text{Name} : N \rightarrow Id, Id \rightarrow N, \text{Addr} : Id \rightarrow A.$
Source and target databases

Source: $Emp(N, A), \ Num(N, Id)$  Target: $Name(Id, N), \ Addr(Id, A)$

Source-to-target dependencies

$\forall n, a. \ Emp(n, a) \Rightarrow \exists id. \ Name(id, n) \land Addr(id, a)$

$\forall n, a, id. \ Emp(n, a) \land Num(n, id) \Rightarrow Name(id, n)$

Target constraints

$Name: \ N \rightarrow Id, Id \rightarrow N, Addr: \ Id \rightarrow A.$

Chase sequence
Source and target databases

Source: \( Emp(N, A), Num(N, Id) \)  
Target: \( Name(Id, N), Addr(Id, A) \)

Source-to-target dependencies

\( \forall n, a. \ Emp(n, a) \Rightarrow \exists id. \ Name(id, n) \land Addr(id, a) \)
\( \forall n, a, id. \ Emp(n, a) \land Num(n, id) \Rightarrow Name(id, n) \)

Target constraints

\( Name : \ N \rightarrow Id, Id \rightarrow N, Addr : \ Id \rightarrow A. \)

Chase sequence

\( I_0 = \{ Emp(Li, LA), Num(Li, 111) \} \)
### Source and target databases

**Source:** \( \text{Emp}(N, A), \text{Num}(N, Id) \)  
**Target:** \( \text{Name}(Id, N), \text{Addr}(Id, A) \)

### Source-to-target dependencies

\( \forall n, a. \text{Emp}(n, a) \Rightarrow \exists id. \text{Name}(id, n) \land \text{Addr}(id, a) \)

\( \forall n, a, id. \text{Emp}(n, a) \land \text{Num}(n, id) \Rightarrow \text{Name}(id, n) \)

### Target constraints

\( \text{Name} : N \rightarrow Id, Id \rightarrow N, \text{Addr} : Id \rightarrow A. \)

### Chase sequence

\( I_0 = \{ \text{Emp}(Li, LA), \text{Num}(Li, 111) \} \)

\( I_1 = \{ \text{Emp}(Li, LA), \text{Num}(Li, 111), \text{Name}(id_1, Li), \text{Addr}(id_1, LA) \} \)
Chase at work

Source and target databases

Source: \( \text{Emp}(N, A), \text{Num}(N, Id) \)  \hspace{1cm} Target: \( \text{Name}(Id, N), \text{Addr}(Id, A) \)

Source-to-target dependencies

\( \forall n, a. \text{Emp}(n, a) \Rightarrow \exists id. \text{Name}(id, n) \land \text{Addr}(id, a) \)

\( \forall n, a, id. \text{Emp}(n, a) \land \text{Num}(n, id) \Rightarrow \text{Name}(id, n) \)

Target constraints

\( \text{Name} : N \rightarrow Id, Id \rightarrow N, \text{Addr} : Id \rightarrow A. \)

Chase sequence

\( I_0 = \{ \text{Emp}(Li, LA), \text{Num}(Li, 111) \} \)

\( I_1 = \{ \text{Emp}(Li, LA), \text{Num}(Li, 111), \text{Name}(id_1, Li), \text{Addr}(id_1, LA) \} \)

\( I_2 = \{ \text{Emp}(Li, LA), \text{Num}(Li, 111), \text{Name}(id_1, Li), \text{Addr}(id_1, LA), \text{Name}(111, Li) \} \)
Chase at work

Source and target databases

Source: $Emp(N, A), Num(N, Id)$  
Target: $Name(Id, N), Addr(Id, A)$

Source-to-target dependencies

$\forall n, a. \ Emp(n, a) \Rightarrow \exists id. \ Name(id, n) \land Addr(id, a)$

$\forall n, a, id. \ Emp(n, a) \land Num(n, id) \Rightarrow Name(id, n)$

Target constraints

$Name: \ N \rightarrow Id, Id \rightarrow N, Addr: \ Id \rightarrow A.$

Chase sequence

$I_0 = \{Emp(Li, LA), Num(Li, 111)\}$

$I_1 = \{Emp(Li, LA), Num(Li, 111), Name(id_1, Li), Addr(id_1, LA)\}$

$I_2 = \{Emp(Li, LA), Num(Li, 111), Name(id_1, Li), Addr(id_1, LA), Name(111, Li)\}$

$I_3 = \{Emp(Li, LA), Num(Li, 111), Name(111, Li), Addr(111, LA)\}$
Chase termination

There is a sequence of chase applications that ends in failure: no universal solution otherwise: every finite sequence that cannot be extended yields a universal solution.

Termination

For weakly acyclic tgds, each chase sequence is of length polynomial in the size of the input.

Data complexity of computing certain answers in PTIME for unions of conjunctive queries (without inequalities) and constraints that are egds and weakly acyclic tgds co-NP-complete for unions of conjunctive queries (with inequalities) and constraints that are egds and weakly acyclic tgds.
Chase termination

Chase result

- there is a sequence of chase applications that ends in failure: no universal solution
- otherwise: every finite sequence that cannot be extended yields a universal solution
Chase result

- there is a sequence of chase applications that ends in failure: no universal solution
- otherwise: every finite sequence that cannot be extended yields a universal solution

Termination

For weakly acyclic tgds, each chase sequence is of length polynomial in the size of the input.
Chase termination

Chase result
- there is a sequence of chase applications that ends in failure: no universal solution
- otherwise: every finite sequence that cannot be extended yields a universal solution

Termination
For weakly acyclic tgds, each chase sequence is of length polynomial in the size of the input.

Data complexity of computing certain answers
- in PTIME for unions of conjunctive queries (without inequalities) and constraints that are egds and weakly acyclic tgds
- co-NP-complete for unions of conjunctive queries (with inequalities) and constraints that are egds and weakly acyclic tgds
Application: semantic query optimization

Rewritings enabled by satisfaction of integrity constraints:
- join elimination/introduction
- predicate elimination/introduction
- eliminating redundancies
Query optimization

- rewrite-based
- cost-based
Application: semantic query optimization

Query optimization
- rewrite-based
- cost-based

Semantic query optimization
Rewritings enabled by satisfaction of integrity constraints:
- join elimination/introduction
- predicate elimination/introduction
- eliminating redundancies
- ...

Jan Chomicki
Preference queries

The winnow operator $\omega_C$ (Chomicki [Cho03])

Find the best answers to a query, according to a given preference relation $\succ_C$.

Relation Book(Title, Vendor, Price)

Preference: $\left(i, v, p \right) \succ_C \left(i', v', p' \right) \equiv i = i' \land p < p'$

Indifference: $\left(i, v, p \right) \sim_C \left(i', v', p' \right) \equiv i \neq i' \lor p = p'$
Preference queries

The winnow operator \( \omega_C \) (Chomicki [Cho03])

Find the best answers to a query, according to a given preference relation \( \succ_C \).

Relation

Book(Title, Vendor, Price)

Preference:

\((i, v, p) \succ_C (i', v', p') \equiv i = i' \land p < p'\)

Indifference:

\((i, v, p) \sim_C (i', v', p') \equiv i \neq i' \lor p = p'\)
Preference queries

The winnow operator $\omega_C$ (Chomicki [Cho03])

Find the best answers to a query, according to a given preference relation $\succ_C$.

Relation $Book(Title, Vendor, Price)$

Preference: $(i, v, p) \succ_C (i', v', p') \equiv i = i' \land p < p'$

Indifference: $(i, v, p) \sim_C (i', v', p') \equiv i \neq i' \lor p = p'$
Preference queries

The winnow operator $\omega_C$ (Chomicki [Cho03])

Find the best answers to a query, according to a given preference relation $\succ_C$.

Relation $Book(Title, Vendor, Price)$

Preference: $(i, v, p) \succ_C (i', v', p') \equiv i = i' \land p < p'$

Indifference: $(i, v, p) \sim_C (i', v', p') \equiv i \neq i' \lor p = p'$

<table>
<thead>
<tr>
<th>Book</th>
<th>Title</th>
<th>Vendor</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>The Flanders Panel</td>
<td>amazon.com</td>
<td>$14.75$</td>
</tr>
<tr>
<td>$t_2$</td>
<td>The Flanders Panel</td>
<td>fatbrain.com</td>
<td>$13.50$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>The Flanders Panel</td>
<td>bn.com</td>
<td>$18.80$</td>
</tr>
<tr>
<td>$t_4$</td>
<td>Green Guide: Greece</td>
<td>bn.com</td>
<td>$17.30$</td>
</tr>
</tbody>
</table>
The winnow operator $\omega_C$ (Chomicki [Cho03])

Find the best answers to a query, according to a given preference relation $\succ C$.

Relation $Book(Title, Vendor, Price)$

Preference: $(i, v, p) \succ C_1 (i', v', p') \equiv i = i' \land p < p'$

Indifference: $(i, v, p) \sim C_1 (i', v', p') \equiv i \neq i' \lor p = p'$

<table>
<thead>
<tr>
<th>Book</th>
<th>Title</th>
<th>Vendor</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>The Flanders Panel</td>
<td>amazon.com</td>
<td>$14.75</td>
</tr>
<tr>
<td>$t_2$</td>
<td>The Flanders Panel</td>
<td>fatbrain.com</td>
<td>$13.50</td>
</tr>
<tr>
<td>$t_3$</td>
<td>The Flanders Panel</td>
<td>bn.com</td>
<td>$18.80</td>
</tr>
<tr>
<td>$t_4$</td>
<td>Green Guide: Greece</td>
<td>bn.com</td>
<td>$17.30</td>
</tr>
</tbody>
</table>
Eliminating redundant occurrences of winnow

Redundant winnow (Chomicki [Cho07b])

Given a set of integrity constraints $\Sigma$, $\omega_C(r) = r$ for every relation $r$ satisfying $\Sigma$ iff $\Sigma$ implies the dependency

$$R(t_1) \land R(t_2) \Rightarrow t_1 \sim_C t_2.$$

Example

Book($i_1, v_1, p_1$) $\land$ Book($i_2, v_2, p_2$) $\Rightarrow$ $i_1 \neq i_2 \lor p_1 = p_2$.

If this dependency is implied by $\Sigma$, $\omega_C(\text{Book}) = \text{Book}$.

Constraint-generating dependencies (Baudinet et al. [BCW95])

general form:

$$\forall t_1, \ldots, t_n. R(t_1) \land \cdots \land R(t_n) \land C(t_1, \ldots, t_n) \Rightarrow C_0(t_1, \ldots, t_n).$$

implication of CGDs is decidable for decidable constraint classes

implication in PTIME for some classes of CGDs

axiomatization not known
Eliminating redundant occurrences of winnow

Redundant winnow (Chomicki [Cho07b])

Given a set of integrity constraints $\Sigma$, $\omega_C(r) = r$ for every relation $r$ satisfying $\Sigma$ iff $\Sigma$ implies the dependency $R(t_1) \land R(t_2) \Rightarrow t_1 \sim_C t_2$. 

Example

Book($i_1, v_1, p_1$) \land Book($i_2, v_2, p_2$) \Rightarrow i_1 \neq i_2 \lor p_1 = p_2.

If this dependency is implied by $\Sigma$, $\omega_C(\text{Book}) = \text{Book}$. 

Constraint-generating dependencies (Baudinet et al. [BCW95])

General form:

$\forall t_1, \ldots, t_n. R(t_1) \land \cdots \land R(t_n) \land C(t_1, \ldots, t_n) \Rightarrow C_0(t_1, \ldots, t_n)$

Implication of CGDs is decidable for decidable constraint classes, implication in PTIME for some classes of CGDs, axiomatization not known.
Eliminating redundant occurrences of winnow

Redundant winnow (Chomicki [Cho07b])

Given a set of integrity constraints $\Sigma$, $\omega_C(r) = r$ for every relation $r$ satisfying $\Sigma$ iff $\Sigma$ implies the dependency $R(t_1) \land R(t_2) \Rightarrow t_1 \sim_C t_2$.

Example

$$Book(i_1, v_1, p_1) \land Book(i_2, v_2, p_2) \Rightarrow i_1 \neq i_2 \lor p_1 = p_2$$

is a functional dependency in disguise:

$$Book(i_1, v_1, p_1) \land Book(i_2, v_2, p_2) \land i_1 = i_2 \Rightarrow p_1 = p_2.$$ 

If this dependency is implied by $\Sigma$, $\omega_C(Book) = Book$. 
Eliminating redundant occurrences of winnow

Redundant winnow (Chomicki [Cho07b])

Given a set of integrity constraints \( \Sigma \), \( \omega_C(r) = r \) for every relation \( r \) satisfying \( \Sigma \) iff \( \Sigma \) implies the dependency \( R(t_1) \land R(t_2) \Rightarrow t_1 \sim_C t_2 \).

Example

\[ Book(i_1, v_1, p_1) \land Book(i_2, v_2, p_2) \Rightarrow i_1 \neq i_2 \lor p_1 = p_2 \]

is a functional dependency in disguise:

\[ Book(i_1, v_1, p_1) \land Book(i_2, v_2, p_2) \land i_1 = i_2 \Rightarrow p_1 = p_2. \]

If this dependency is implied by \( \Sigma \), \( \omega_C(Book) = Book \).

Constraint-generating dependencies (Baudinet et al. [BCW95])

- general form: \( \forall t_1, \ldots, t_n. \ R(t_1) \land \cdots \land R(t_n) \land C(t_1, \ldots, t_n) \Rightarrow C_0(t_1, \ldots, t_n) \)
- implication of CGDs is decidable for decidable constraint classes
- implication in PTIME for some classes of CGDs
- axiomatization not known
Part II

Consistent query answers
Sources of inconsistency:

- integration of independent data sources with overlapping data
- time lag of updates (eventual consistency)
- unenforced integrity constraints
- dataspace systems,...
Whence Inconsistency?

Sources of inconsistency:
- integration of independent data sources with overlapping data
- time lag of updates (eventual consistency)
- unenforced integrity constraints
- dataspace systems,...

Eliminating inconsistency?
- not enough information, time, or money
- difficult, impossible or undesirable
- unnecessary: queries may be insensitive to inconsistency
Query results not reliable.
Query results **not reliable**.

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name → City Salary
Query results *not reliable*.

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

SELECT Name
FROM Employee
WHERE Salary ≤ 25M

Name → City Salary
Ignoring Inconsistency

Query results not reliable.

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

SELECT Name
FROM Employee
WHERE Salary ≤ 25M
Horizontal Decomposition

Decomposition into two relations:

- violators
- the rest

(De Bra, Paredaens [DBP83])
Horizontal Decomposition

Decomposition into two relations:
- violators
- the rest

(De Bra, Paredaens [DBP83])

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name → City Salary
Horizontal Decomposition

Decomposition into two relations:
- violators
- the rest

(De Bra, Paredaens [DBP83])

Name | City     | Salary
-----|----------|--------
Gates| Redmond  | 20M    
Gates| Redmond  | 30M    
Grove| Santa Clara| 10M   

Name → City Salary

Grove | Santa Clara | 10M
Name → City Salary

Gates | Redmond | 20M
Gates | Redmond | 30M
Name → City Salary
Exceptions to Constraints

Weakening the constraints:
- functional dependencies $\rightarrow$ denial constraints

(Borgida [Bor85])
Weakening the constraints:

- functional dependencies $\rightarrow$ denial constraints

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name $\rightarrow$ City Salary

(Borgida [Bor85])
Exceptions to Constraints

Weakening the constraints:
- functional dependencies $\rightarrow$ denial constraints

(Borgida [Bor85])

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name $\rightarrow$ City Salary

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name $\rightarrow$ City Salary except Name='Gates'
The Impact of Inconsistency on Queries

Traditional view

- query results defined irrespective of integrity constraints
- query evaluation may be optimized in the presence of integrity constraints (semantic query optimization)
The Impact of Inconsistency on Queries

**Traditional view**
- query results defined irrespective of integrity constraints
- query evaluation may be optimized in the presence of integrity constraints (semantic query optimization)

**Our view**
- inconsistency reflects **uncertainty**
- query results may depend on integrity constraint satisfaction
- inconsistency may be eliminated or tolerated
Restoring consistency:

- insertion, deletion, update
- minimal change?
Database Repairs

### Restoring consistency:

- insertion, deletion, update
- minimal change?

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name → City Salary
Database Repairs

Restoring consistency:
- insertion, deletion, update
- minimal change?

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name → City Salary

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name → City Salary
Consistent query answer:
Query answer obtained in every repair.

(Arenas, Bertossi, Chomicki [ABC99])
Consistent Query Answering

Consistent query answer:
Query answer obtained in every repair.

(Arenas, Bertossi, Chomicki [ABC99])

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name → City Salary
Consistent query answer:

Query answer obtained in every repair.

(Arenas, Bertossi, Chomicki [ABC99])

SELECT Name
FROM Employee
WHERE Salary ≤ 25M

Name → City Salary

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>
Consistent query answer:
Query answer obtained in every repair.

(Arenas, Bertossi, Chomicki [ABC99])

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

SELECT Name
FROM Employee
WHERE Salary ≥ 10M
Research Goals

Formal definition

What constitutes reliable (consistent) information in an inconsistent database.
Research Goals

Formal definition
What constitutes reliable (consistent) information in an inconsistent database.

Algorithms
How to compute consistent information.
Research Goals

Formal definition
What constitutes reliable (consistent) information in an inconsistent database.

Algorithms
How to compute consistent information.

Computational complexity analysis
- tractable vs. intractable classes of queries and integrity constraints
- tradeoffs: complexity vs. expressiveness.
Research Goals

Formal definition
What constitutes reliable (consistent) information in an inconsistent database.

Algorithms
How to compute consistent information.

Computational complexity analysis
- tractable vs. intractable classes of queries and integrity constraints
- tradeoffs: complexity vs. expressiveness.

Implementation
- preferably using DBMS technology.
Research Goals

Formal definition

What constitutes reliable (consistent) information in an inconsistent database.

Algorithms

How to compute consistent information.

Computational complexity analysis

- tractable vs. intractable classes of queries and integrity constraints
- tradeoffs: complexity vs. expressiveness.

Implementation

- preferably using DBMS technology.

Applications

???
Repair $D'$ of a database $D$ w.r.t. the integrity constraints $IC$:

- $D'$: over the same schema as $D$
- $D' \models IC$
- symmetric difference between $D$ and $D'$ is minimal.
Repair $D'$ of a database $D$ w.r.t. the integrity constraints $IC$:

- $D'$: over the same schema as $D$
- $D' \models IC$
- symmetric difference between $D$ and $D'$ is minimal.

Consistent query answer to a query $Q$ in $D$ w.r.t. $IC$:

- an element of the result of $Q$ in every repair of $D$ w.r.t. $IC$. 

Another incarnation of the idea of sure query answers [Lipski: TODS'79].
**Basic Notions**

**Repair** $D'$ of a database $D$ w.r.t. the integrity constraints $IC$:

- $D'$: over the same schema as $D$
- $D' \models IC$
- symmetric difference between $D$ and $D'$ is **minimal**.

**Consistent query answer to a query $Q$ in $D$ w.r.t. $IC$**:

- an element of the result of $Q$ in **every repair** of $D$ w.r.t. $IC$.

Another incarnation of the idea of **sure** query answers [Lipski: TODS’79].
Belief revision

- semantically: repairing $\equiv$ revising the database with integrity constraints
- consistent query answers $\equiv$ counterfactual inference.

Logical inconsistency

- inconsistent database: database facts together with integrity constraints form an inconsistent set of formulas
- trivialization of reasoning does not occur because constraints are not used in relational query evaluation.
Exponentially many repairs

Example relation $R(A, B)$

- violates the dependency $A \rightarrow B$
- has $2^n$ repairs.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$a_n$</td>
<td>$b_n$</td>
</tr>
<tr>
<td>$a_n$</td>
<td>$c_n$</td>
</tr>
</tbody>
</table>

$A \rightarrow B$
Exponentially many repairs

Example relation $R(A, B)$

- violates the dependency $A \rightarrow B$
- has $2^n$ repairs.

\[
\begin{array}{|c|c|}
\hline
A & B \\
\hline
a_1 & b_1 \\
a_1 & c_1 \\
a_2 & b_2 \\
a_2 & c_2 \\
\vdots \\
a_n & b_n \\
a_n & c_n \\
\hline
\end{array}
\]

$A \rightarrow B$

It is impractical to apply the definition of CQA directly.
Query Rewriting

Given a query $Q$ and a set of integrity constraints $IC$, build a query $Q^{IC}$ such that for every database instance $D$

\[
\text{the set of answers to } Q^{IC} \text{ in } D = \text{the set of consistent answers to } Q \text{ in } D \text{ w.r.t. } IC.
\]
## Computing Consistent Query Answers

### Query Rewriting

Given a query $Q$ and a set of integrity constraints $IC$, build a query $Q^{IC}$ such that for every database instance $D$

\[
\text{the set of answers to } Q^{IC} \text{ in } D = \text{the set of consistent answers to } Q \text{ in } D \text{ w.r.t. } IC.
\]

### Representing all repairs

Given $IC$ and $D$:

1. build a space-efficient representation of all repairs of $D$ w.r.t. $IC$
2. use this representation to answer (many) queries.
Computing Consistent Query Answers

Query Rewriting

Given a query $Q$ and a set of integrity constraints $IC$, build a query $Q^{IC}$ such that for every database instance $D$

\[ \text{the set of answers to } Q^{IC} \text{ in } D = \text{the set of consistent answers to } Q \text{ in } D \text{ w.r.t. } IC. \]

Representing all repairs

Given $IC$ and $D$:

1. build a space-efficient representation of all repairs of $D$ w.r.t. $IC$
2. use this representation to answer (many) queries.

Logic programs

Given $IC$, $D$ and $Q$:

1. build a logic program $P_{IC,D}$ whose models are the repairs of $D$ w.r.t. $IC$
2. build a logic program $P_Q$ expressing $Q$
3. use a logic programming system that computes the query atoms present in all models of $P_{IC,D} \cup P_Q$. 
Universal constraints

∀. ¬A_1 ∨ ⋯ ∨ ¬A_n ∨ B_1 ∨ ⋯ ∨ B_m
### Universal constraints
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m \]

### Example
\[ \forall. \neg Par(x) \lor Ma(x) \lor Fa(x) \]
Constraint classes

### Universal constraints

\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m \]

### Example

\[ \forall. \neg Par(x) \lor Ma(x) \lor Fa(x) \]

### Denial constraints

\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \]
## Constraint classes

### Universal constraints

\[ \forall \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m \]

**Example**

\[ \forall \neg Par(x) \lor Ma(x) \lor Fa(x) \]

### Denial constraints

\[ \forall \neg A_1 \lor \cdots \lor \neg A_n \]

**Example**

\[ \forall \neg M(n, s, m) \lor \neg M(m, t, w) \lor s \leq t \]
Constraint classes

Universal constraints
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m \]

Example
\[ \forall. \neg Par(x) \lor Ma(x) \lor Fa(x) \]

Denial constraints
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \]

Example
\[ \forall. \neg M(n, s, m) \lor \neg M(m, t, w) \lor s \leq t \]

Functional dependencies
\[ X \rightarrow Y: \]
- a key dependency in \( F \) if \( Y = U \)
- a primary-key dependency: only one key exists
### Constraint classes

<table>
<thead>
<tr>
<th>Universal constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall. \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall. \neg Par(x) \lor Ma(x) \lor Fa(x)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Denial constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall. \neg A_1 \lor \cdots \lor \neg A_n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall. \neg M(n, s, m) \lor \neg M(m, t, w) \lor s \leq t$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functional dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \rightarrow Y$:</td>
</tr>
<tr>
<td>- a <strong>key</strong> dependency in $F$ if $Y = U$</td>
</tr>
<tr>
<td>- a <strong>primary-key</strong> dependency: only one key exists</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example primary-key dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name $\rightarrow$ Address Salary</td>
</tr>
</tbody>
</table>
**Constraint classes**

### Universal constraints

\[
\forall. \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m
\]

**Example**

\[
\forall. \neg Par(x) \lor Ma(x) \lor Fa(x)
\]

### Denial constraints

\[
\forall. \neg A_1 \lor \cdots \lor \neg A_n
\]

**Example**

\[
\forall. \neg M(n, s, m) \lor \neg M(m, t, w) \lor s \leq t
\]

### Functional dependencies

\[X \rightarrow Y:\]

- a key dependency in \(F\) if \(Y = U\)
- a primary-key dependency: only one key exists

**Example primary-key dependency**

Name → Address Salary

### Inclusion dependencies

\(R[X] \subseteq S[Y] :\)

- a foreign key constraint if \(Y\) is a key of \(S\)
### Constraint classes

#### Universal constraints
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m \]

**Example**
\[ \forall. \neg Par(x) \lor Ma(x) \lor Fa(x) \]

#### Denial constraints
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \]

**Example**
\[ \forall. \neg M(n, s, m) \lor \neg M(m, t, w) \lor s \leq t \]

#### Functional dependencies
\[ X \rightarrow Y: \]
- a **key** dependency in \( F \) if \( Y = U \)
- a **primary-key** dependency: only one key exists

**Example primary-key dependency**
Name \( \rightarrow \) Address Salary

#### Inclusion dependencies
\[ R[X] \subseteq S[Y]: \]
- a **foreign key** constraint if \( Y \) is a key of \( S \)

**Example foreign key constraint**
\( M[Manager] \subseteq M[Name] \)
Building queries that compute CQAs

- relational calculus (algebra) $\sim$ relational calculus (algebra)
- SQL $\sim$ SQL
- leads to PTIME data complexity
Building queries that compute CQAs

- relational calculus (algebra) $\leadsto$ relational calculus (algebra)
- SQL $\leadsto$ SQL
- leads to PTIME data complexity

Query

Emp(x, y, z)
Query Rewriting

Building queries that compute CQAs

- relational calculus (algebra) $\Rightarrow$ relational calculus (algebra)
- SQL $\Rightarrow$ SQL
- leads to PTIME data complexity

Query

$Emp(x, y, z)$

Integrity constraint

$\forall x, y, z, y', z'. \neg Emp(x, y, z) \lor \neg Emp(x, y', z') \lor z = z'$
Query Rewriting

Building queries that compute CQAs

- relational calculus (algebra) \(\sim\) relational calculus (algebra)
- SQL \(\sim\) SQL
- leads to PTIME data complexity

Query

\[ \text{Emp}(x, y, z) \]

Integrity constraint

\[
\forall x, y, z, y', z'. \quad \neg \text{Emp}(x, y, z) \lor \neg \text{Emp}(x, y', z') \lor z = z'
\]
Query Rewriting

Building queries that compute CQAs

- relational calculus (algebra) $\sim$ relational calculus (algebra)
- SQL $\sim$ SQL
- leads to PTIME data complexity

Query

$Emp(x, y, z)$

Integrity constraint

$\forall x, y, z, y', z'. \neg Emp(x, y, z) \lor \neg Emp(x, y', z') \lor z = z'$

Rewritten query

$Emp(x, y, z) \land \forall y', z'. \neg Emp(x, y', z') \lor z = z'$
(Arenas, Bertossi, Chomicki [ABC99])

- Queries: *conjunctions* of literals (relational algebra: $\sigma, \times, -$)
- Integrity constraints: *binary universal*
The Scope of Query Rewriting

(Arenas, Bertossi, Chomicki [ABC99])

- Queries: conjunctions of literals (relational algebra: \( \sigma, \times, - \))
- Integrity constraints: binary universal

(Fuxman, Miller [FM05b])

- Queries: \( C_{forest} \)
  - a class of conjunctive queries (\( \pi, \sigma, \times \))
  - no non-key or non-full joins
  - no repeated relation symbols
  - no built-ins
- Integrity constraints: primary key functional dependencies
SQL query

```sql
SELECT Name FROM Emp
WHERE Salary ≥ 10K
```
SQL query

```sql
SELECT Name FROM Emp
WHERE Salary ≥ 10K
```

SQL rewritten query

```sql
SELECT e1.Name FROM Emp e1
WHERE e1.Salary ≥ 10K AND NOT EXISTS
  (SELECT * FROM EMPLOYEE e2
   WHERE e2.Name = e1.Name AND e2.Salary < 10K)
```
SQL Rewriting

**SQL query**

```
SELECT Name FROM Emp
WHERE Salary ≥ 10K
```

**SQL rewritten query**

```
SELECT e1.Name FROM Emp e1
WHERE e1.Salary ≥ 10K AND NOT EXISTS
  (SELECT * FROM EMPLOYEE e2
   WHERE e2.Name = e1.Name AND e2.Salary < 10K)
```

(Fuxman, Fazli, Miller [FM05a])

- **ConQuer**: a system for computing CQAs
- conjunctive ($C_{forest}$) and aggregation SQL queries
- databases can be annotated with consistency indicators
- tested on TPC-H queries and medium-size databases
Conflict Hypergraph

Vertices
Tuples in the database.

(Gates, Redmond, 20M)

(Grove, Santa Clara, 10M)

(Gates, Redmond, 30M)
Conflict Hypergraph

**Vertices**
Tuples in the database.

**Edges**
Minimal sets of tuples violating a constraint.

(Gates, Redmond, 20M)
(Grove, Santa Clara, 10M)
(Gates, Redmond, 30M)
Conflict Hypergraph

**Vertices**
Tuples in the database.

**Edges**
Minimal sets of tuples violating a constraint.

**Repairs**
Maximal independent sets in the conflict graph.

- (Gates, Redmond, 20M)
- (Grove, Santa Clara, 10M)
- (Gates, Redmond, 30M)
Conflict Hypergraph

**Vertices**

Tuples in the database.

**Edges**

Minimal sets of tuples violating a constraint.

**Repairs**

Maximal independent sets in the conflict graph.

- (Gates, Redmond, 20M)
- (Grove, Santa Clara, 10M)
- (Gates, Redmond, 30M)
Algorithm HProver

INPUT: query $\Phi$ a disjunction of ground atoms, conflict hypergraph $G$
OUTPUT: is $\Phi$ false in some repair of $D$ w.r.t. $IC$?

ALGORITHM:

1. $\neg \Phi = P_1(t_1) \land \cdots \land P_m(t_m) \land \neg P_{m+1}(t_{m+1}) \land \cdots \land \neg P_n(t_n)$

2. find a consistent set of facts $S$ such that

   - $S \supseteq \{P_1(t_1), \ldots, P_m(t_m)\}$
   - for every fact $A \in \{P_{m+1}(t_{m+1}), \ldots, P_n(t_n)\}$: $A \not\in D$ or there is an edge $E = \{A, B_1, \ldots, B_m\}$ in $G$ and $S \supseteq \{B_1, \ldots, B_m\}$.
Computing CQAs Using Conflict Hypergraphs

**Algorithm HProver**

**INPUT:** query Φ a disjunction of ground atoms, conflict hypergraph G

**OUTPUT:** is Φ false in some repair of D w.r.t. IC?

**ALGORITHM:**

1. \( \neg \Phi = P_1(t_1) \land \cdots \land P_m(t_m) \land \neg P_{m+1}(t_{m+1}) \land \cdots \land \neg P_n(t_n) \)

2. find a consistent set of facts \( S \) such that
   - \( S \supseteq \{P_1(t_1), \ldots, P_m(t_m)\} \)
   - for every fact \( A \in \{P_{m+1}(t_{m+1}), \ldots, P_n(t_n)\} \): \( A \not\in D \) or there is an edge \( E = \{A, B_1, \ldots, B_m\} \) in \( G \) and \( S \supseteq \{B_1, \ldots, B_m\} \).

(Chomicki, Marcinkowski, Staworko [CMS04])

- **Hippo:** a system for computing CQAs in PTIME
- quantifier-free queries and denial constraints
- only edges of the conflict hypergraph are kept in main memory
- optimization can eliminate many (sometimes all) database accesses in HProver
- tested for medium-size synthetic databases
Specifying repairs as answer sets of logic programs

- (Arenas, Bertossi, Chomicki [ABC03])
- (Greco, Greco, Zumpano [GGZ03])
- (Calì, Lembo, Rosati [CLR03b])
Specifying repairs as answer sets of logic programs

- (Arenas, Bertossi, Chomicki [ABC03])
- (Greco, Greco, Zumpano [GGZ03])
- (Calì, Lembo, Rosati [CLR03b])

Example

\[
\text{emp}(x, y, z) \leftarrow \text{emp}_D(x, y, z), \text{not dubious_emp}(x, y, z). \\
\text{dubious_emp}(x, y, z) \leftarrow \text{emp}_D(x, y, z), \text{emp}(x, y', z'), y \neq y'. \\
\text{dubious_emp}(x, y, z) \leftarrow \text{emp}_D(x, y, z), \text{emp}(x, y', z'), z \neq z'.
\]
Specifying repairs as answer sets of logic programs

- (Arenas, Bertossi, Chomicki [ABC03])
- (Greco, Greco, Zumpano [GGZ03])
- (Calì, Lembo, Rosati [CLR03b])

Example

\[
\text{emp}(x, y, z) & \leftarrow \text{emp}_D(x, y, z), \text{not dubious\_emp}(x, y, z).
\]
\[
\text{dubious\_emp}(x, y, z) & \leftarrow \text{emp}_D(x, y, z), \text{emp}(x, y', z'), y \neq y'.
\]
\[
\text{dubious\_emp}(x, y, z) & \leftarrow \text{emp}_D(x, y, z), \text{emp}(x, y', z'), z \neq z'.
\]

Answer sets

- \{\text{emp}(Gates, Redmond, 20M), \text{emp}(Grove, SantaClara, 10M), \ldots\}\n- \{\text{emp}(Gates, Redmond, 30M), \text{emp}(Grove, SantaClara, 10M), \ldots\}\
Logic Programs for computing CQAs

Logic Programs

- disjunction and classical negation
- checking whether an atom is in all answer sets is $\Pi_2^P$-complete
- dlv, smodels, ...
Logic Programs

- disjunction and classical negation
- checking whether an atom is in all answer sets is $\Pi_2^P$-complete
- dlv, smodels, ...

Scope

- arbitrary first-order queries
- universal constraints
- approach unlikely to yield tractable cases
Logic Programs for computing CQAs

Logic Programs

- disjunction and classical negation
- checking whether an atom is in all answer sets is $\Pi_2^P$-complete
- dlv, smodels, ...

Scope

- arbitrary first-order queries
- universal constraints
- approach unlikely to yield tractable cases

INFOMIX (Eiter et al. [EFGL03])

- combines CQA with data integration (GAV)
- uses dlv for repair computations
- optimization techniques: localization, factorization
- tested on small-to-medium-size legacy databases
Theorem (Chomicki, Marcinkowski [CM05a])

For primary-key functional dependencies and conjunctive queries, consistent query answering is data-complete for co-NP.
Co-NP-completeness of CQA

Theorem (Chomicki, Marcinkowski [CM05a])

For primary-key functional dependencies and conjunctive queries, consistent query answering is data-complete for co-NP.

Proof.

Membership: $S$ is a repair iff $S \models IC$ and $W \not\models IC$ if $W = S \cup A$.

Co-NP-hardness: reduction from MONOTONE 3-SAT.

1. Positive clauses $\beta_1 = \phi_1 \land \cdots \land \phi_m$, negative clauses $\beta_2 = \psi_{m+1} \land \cdots \land \psi_l$.
2. Database $D$ contains two binary relations $R(A, B)$ and $S(A, B)$:
   - $R(i, p)$ if variable $p$ occurs in $\phi_i, i = 1, \ldots, m$.
   - $S(i, p)$ if variable $p$ occurs in $\psi_i, i = m+1, \ldots, l$.
3. $A$ is the primary key of both $R$ and $S$.
4. Query $Q \equiv \exists x, y, z. (R(x, y) \land S(z, y))$.
5. There is an assignment which satisfies $\beta_1 \land \beta_2$ iff there exists a repair in which $Q$ is false.
Theorem (Chomicki, Marcinkowski [CM05a])

For primary-key functional dependencies and conjunctive queries, consistent query answering is data-complete for co-NP.

Proof.

Membership: $S$ is a repair iff $S \models IC$ and $W \not\models IC$ if $W = S \cup A$.

Co-NP-hardness: reduction from MONOTONE 3-SAT.

1. Positive clauses $\beta_1 = \phi_1 \land \cdots \land \phi_m$, negative clauses $\beta_2 = \psi_{m+1} \land \cdots \land \psi_l$.

2. Database $D$ contains two binary relations $R(A, B)$ and $S(A, B)$:
   - $R(i, p)$ if variable $p$ occurs in $\phi_i$, $i = 1, \ldots, m$.
   - $S(i, p)$ if variable $p$ occurs in $\psi_i$, $i = m + 1, \ldots, l$.

3. $A$ is the primary key of both $R$ and $S$.

4. Query $Q \equiv \exists x, y, z. (R(x, y) \land S(z, y))$.

5. There is an assignment which satisfies $\beta_1 \land \beta_2$ iff there exists a repair in which $Q$ is false.

$Q$ does not belong to $C_{forest}$. 
## Data complexity of CQA

<table>
<thead>
<tr>
<th></th>
<th>Primary keys</th>
<th>Arbitrary keys</th>
<th>Denial</th>
<th>Universal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma, \times, -$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \times, -, \cup$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi, \times$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi, \times, -, \cup$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Data complexity of CQA

<table>
<thead>
<tr>
<th>Primary keys</th>
<th>Arbitrary keys</th>
<th>Denial</th>
<th>Universal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma, \times, -$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME: binary</td>
</tr>
<tr>
<td>$\sigma, \times, -, \cup$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi, \times$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi, \times, -, \cup$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Árenas, Bertossi, Chomicki [ABC99])
## Data complexity of CQA

<table>
<thead>
<tr>
<th>Primary keys</th>
<th>Arbitrary keys</th>
<th>Denial</th>
<th>Universal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma, \times, -$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME: binary</td>
</tr>
<tr>
<td>$\sigma, \times, -, U$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
</tr>
<tr>
<td>$\sigma, \pi$</td>
<td>PTIME</td>
<td>co-NPC</td>
<td>co-NPC</td>
</tr>
<tr>
<td>$\sigma, \pi, \times$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
</tr>
<tr>
<td>$\sigma, \pi, \times, -, U$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
</tr>
</tbody>
</table>

- (Arenas, Bertossi, Chomicki [ABC99])
- (Chomicki, Marcinkowski [CM05a])
### Data complexity of CQA

<table>
<thead>
<tr>
<th></th>
<th>Primary keys</th>
<th>Arbitrary keys</th>
<th>Denial</th>
<th>Universal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma, \times, -$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME: binary</td>
</tr>
<tr>
<td>$\sigma, \times, -, \cup$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
</tr>
<tr>
<td>$\sigma, \pi$</td>
<td>PTIME</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
</tr>
<tr>
<td>$\sigma, \pi, \times$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
</tr>
<tr>
<td>$\sigma, \pi, \times$, C\text{forest}</td>
<td>PTIME: C\text{forest}</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
</tr>
<tr>
<td>$\sigma, \pi, \times, -, \cup$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
</tr>
</tbody>
</table>

- [Arenas, Bertossi, Chomicki [ABC99]]
- [Chomicki, Marcinkowski [CM05a]]
- [Fuxman, Miller [FM05b]]
Data complexity of CQA

<table>
<thead>
<tr>
<th></th>
<th>Primary keys</th>
<th>Arbitrary keys</th>
<th>Denial</th>
<th>Universal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma, \times, -$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME: binary $\Pi_2^p$-complete</td>
</tr>
<tr>
<td>$\sigma, \times, -, \cup$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td>$\Pi_2^p$-complete</td>
</tr>
<tr>
<td>$\sigma, \pi$</td>
<td>PTIME</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>$\Pi_2^p$-complete</td>
</tr>
<tr>
<td>$\sigma, \pi, \times$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>$\Pi_2^p$-complete</td>
</tr>
<tr>
<td>$\sigma, \pi, \times, \cup$</td>
<td>PTIME: $C_{forest}$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>$\Pi_2^p$-complete</td>
</tr>
</tbody>
</table>

- (Arenas, Bertossi, Chomicki [ABC99])
- (Chomicki, Marcinkowski [CM05a])
- (Fuxman, Miller [FM05b])
- (Staworko, Ph.D., 2007)
**Tuple-based repairs**

- asymmetric treatment of insertion and deletion:
  - repairs by minimal deletions only (Chomicki, Marcinkowski [CM05a]): data possibly **incorrect** but **complete**
  - repairs by minimal deletions and arbitrary insertions (Calì, Lembo, Rosati [CLR03a]): data possibly **incorrect** and **incomplete**

- minimal cardinality changes (Lopatenko, Bertossi [LB07])
The Semantic Explosion

Tuple-based repairs

- asymmetric treatment of insertion and deletion:
  - repairs by minimal deletions only (Chomicki, Marcinkowski [CM05a]): data possibly incorrect but complete
  - repairs by minimal deletions and arbitrary insertions (Calì, Lembo, Rosati [CLR03a]): data possibly incorrect and incomplete
- minimal cardinality changes (Lopatenko, Bertossi [LB07])

Attribute-based repairs

- (A) ground and non-ground repairs (Wijsen [Wij05])
- (B) project-join repairs (Wijsen [Wij06])
- (C) repairs minimizing Euclidean distance (Bertossi et al. [BBFL05])
- (D) repairs of minimum cost (Bohannon et al. [BFFR05])
The Semantic Explosion

Tuple-based repairs

- asymmetric treatment of insertion and deletion:
  - repairs by minimal deletions only (Chomicki, Marcinkowski [CM05a]): data possibly incorrect but complete
  - repairs by minimal deletions and arbitrary insertions (Calì, Lembo, Rosati [CLR03a]): data possibly incorrect and incomplete
- minimal cardinality changes (Lopatenko, Bertossi [LB07])

Attribute-based repairs

- (A) ground and non-ground repairs (Wijsen [Wij05])
- (B) project-join repairs (Wijsen [Wij06])
- (C) repairs minimizing Euclidean distance (Bertossi et al. [BBFL05])
- (D) repairs of minimum cost (Bohannon et al. [BFFR05])

Computational complexity

- (A) and (B): similar to tuple based repairs
- (C) and (D): checking existence of a repair of cost $< K$ NP-complete.
The Need for Attribute-based Repairing

Tuple-based repairing leads to information loss.
The Need for Attribute-based Repairing

Tuple-based repairing leads to **information loss**.

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Sales</td>
<td>Buffalo</td>
</tr>
<tr>
<td>Mary</td>
<td>Sales</td>
<td>Toronto</td>
</tr>
</tbody>
</table>

_Name_ → _Dept_

_Dept_ → _City_
Tuple-based repairing leads to information loss.

**EmpDept**

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Sales</td>
<td>Buffalo</td>
</tr>
<tr>
<td>Mary</td>
<td>Sales</td>
<td>Toronto</td>
</tr>
</tbody>
</table>

Name $\rightarrow$ Dept
Dept $\rightarrow$ City
Repair a **lossless join decomposition**.

The decomposition:

\[ \pi_{Name, \text{Dept}}(\text{EmpDept}) \Join \pi_{\text{Dept}, \text{Location}}(\text{EmpDept}) \]
Repair a lossless join decomposition.

The decomposition:

\[ \pi_{\text{Name},\text{Dept}}(\text{EmpDept}) \bowtie \pi_{\text{Dept},\text{Location}}(\text{EmpDept}) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Sales</td>
<td>Buffalo</td>
</tr>
<tr>
<td>John</td>
<td>Sales</td>
<td>Toronto</td>
</tr>
<tr>
<td>Mary</td>
<td>Sales</td>
<td>Buffalo</td>
</tr>
<tr>
<td>Mary</td>
<td>Sales</td>
<td>Toronto</td>
</tr>
</tbody>
</table>

Name \to Dept

Dept \to City
Repair a **lossless join decomposition**.

The decomposition:

\[ \pi_{\text{Name, Dept}}(\text{EmpDept}) \bowtie \pi_{\text{Dept, Location}}(\text{EmpDept}) \]
Probabilistic framework for “dirty” databases

(Andritsos, Fuxman, Miller [AFM06])

- potential duplicates identified and grouped into clusters
- worlds $\approx$ repairs: one tuple from each cluster
- world probability: product of tuple probabilities
- clean answers: in the query result in some (supporting) world
- clean answer probability: sum of the probabilities of supporting worlds
  - consistent answer: clean answer with probability 1
Probabilistic framework for “dirty” databases

(Andritsos, Fuxman, Miller [AFM06])

- potential duplicates identified and grouped into clusters
- worlds $\approx$ repairs: one tuple from each cluster
- world probability: product of tuple probabilities
- clean answers: in the query result in some (supporting) world
- clean answer probability: sum of the probabilities of supporting worlds
  - consistent answer: clean answer with probability 1

Salaries with probabilities

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>20M</td>
<td>0.7</td>
</tr>
<tr>
<td>Gates</td>
<td>30M</td>
<td>0.3</td>
</tr>
<tr>
<td>Grove</td>
<td>10M</td>
<td>0.5</td>
</tr>
<tr>
<td>Grove</td>
<td>20M</td>
<td>0.5</td>
</tr>
</tbody>
</table>
SQL query

```
SELECT Name
FROM EmpProb e
WHERE e.Salary > 15M
```
SQL query

```
SELECT Name
FROM EmpProb e
WHERE e.Salary > 15M
```

SQL rewritten query

```
SELECT e.Name, SUM(e.Prob)
FROM EmpProb e
WHERE e.Salary > 15M
GROUP BY e.Name
```
Computing Clean Answers

**SQL query**

```sql
SELECT Name
FROM EmpProb e
WHERE e.Salary > 15M
```

**SQL rewritten query**

```sql
SELECT e.Name, SUM(e.Prob)
FROM EmpProb e
WHERE e.Salary > 15M
GROUP BY e.Name
```

**EmpProb**

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>20M</td>
<td>0.7</td>
</tr>
<tr>
<td>Gates</td>
<td>30M</td>
<td>0.3</td>
</tr>
<tr>
<td>Grove</td>
<td>10M</td>
<td>0.5</td>
</tr>
<tr>
<td>Grove</td>
<td>20M</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Name ➔ Salary
### SQL query

```
SELECT Name
FROM EmpProb e
WHERE e.Salary > 15M
```

### SQL rewritten query

```
SELECT e.Name, SUM(e.Prob)
FROM EmpProb e
WHERE e.Salary > 15M
GROUP BY e.Name
```

---

**EmpProb**

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>20M</td>
<td>0.7</td>
</tr>
<tr>
<td>Gates</td>
<td>30M</td>
<td>0.3</td>
</tr>
<tr>
<td>Grove</td>
<td>10M</td>
<td>0.5</td>
</tr>
<tr>
<td>Grove</td>
<td>20M</td>
<td>0.5</td>
</tr>
</tbody>
</table>

**Name → Salary**
Computing Clean Answers

SQL query

```
SELECT Name
FROM EmpProb e
WHERE e.Salary > 15M
```

SQL rewritten query

```
SELECT e.Name, SUM(e.Prob)
FROM EmpProb e
WHERE e.Salary > 15M
GROUP BY e.Name
```

### EmpProb

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>20M</td>
<td>0.7</td>
</tr>
<tr>
<td>Gates</td>
<td>30M</td>
<td>0.3</td>
</tr>
<tr>
<td>Grove</td>
<td>10M</td>
<td>0.5</td>
</tr>
<tr>
<td>Grove</td>
<td>20M</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Name → Salary

```
SELECT e.Name, SUM(e.Prob)
FROM EmpProb e
WHERE e.Salary > 15M
GROUP BY e.Name
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>1</td>
</tr>
<tr>
<td>Grove</td>
<td>0.5</td>
</tr>
</tbody>
</table>
Technology

- **practical methods** for CQA for a subset of SQL:
  - restricted conjunctive/aggregation queries, primary/foreign-key constraints
  - quantifier-free queries/denial constraints
  - LP-based approaches for expressive query/constraint languages
- implemented in **prototype systems**
- tested on **medium-size databases**
Taking Stock: Good News

Technology

- **practical methods** for CQA for a subset of SQL:
  - restricted conjunctive/aggregation queries, primary/foreign-key constraints
  - quantifier-free queries/denial constraints
  - LP-based approaches for expressive query/constraint languages
- implemented in **prototype systems**
- tested on **medium-size databases**

The CQA Community

- over 30 active researchers
- around 100 publications (since 1999)
- outreach to the AI community (qualified success)
- overview papers [BC03, Ber06, Cho07a, CM05b]
"Blending in" CQA data integration: tension between repairing and satisfying source-to-target dependencies peer-to-peer: how to isolate an inconsistent peer?

Extensions

nulls: repairs with nulls? clean semantics vs. SQL conformance priorities: preferred repairs application: conflict resolution

XML notions of integrity constraint and repair repair minimality based on tree edit distance?

aggregate constraints

Jan Chomicki () Database Consistency
“Blending in” CQA

- **data integration**: tension between repairing and satisfying source-to-target dependencies
- **peer-to-peer**: how to isolate an inconsistent peer?

Extensions

- nulls: repairs with nulls?
- clean semantics vs. SQL conformance
- preferred repairs
- application: conflict resolution

XML notions of integrity constraint and repair

- repair minimality based on tree edit distance?

- aggregate constraints
Taking Stock: Initial Progress

“Blending in” CQA

- **data integration**: tension between repairing and satisfying source-to-target dependencies
- **peer-to-peer**: how to isolate an inconsistent peer?

Extensions

- **nulls**:
  - repairs with nulls?
  - clean semantics vs. SQL conformance
- **priorities**:
  - preferred repairs
  - application: conflict resolution
- **XML**
  - notions of integrity constraint and repair
  - repair minimality based on tree edit distance?
- **aggregate constraints**
Taking Stock: Largely Open Issues

Applications

- no deployed applications
- repairing vs. CQA: data and query characteristics
- heuristics for CQA and repairing
### Applications
- no deployed applications
- repairing vs. CQA: data and query characteristics
- heuristics for CQA and repairing

### CQA in context
- taming the semantic explosion
- CQA and data cleaning
- CQA and schema matching/mapping
Taking Stock: Largely Open Issues

Applications
- no deployed applications
- repairing vs. CQA: data and query characteristics
- heuristics for CQA and repairing

CQA in context
- taming the semantic explosion
- CQA and data cleaning
- CQA and schema matching/mapping

Foundations
- defining measures of consistency
- more refined complexity analysis, dynamic aspects
### Applications
- no deployed applications
- repairing vs. CQA: data and query characteristics
- heuristics for CQA and repairing

### CQA in context
- taming the semantic explosion
- CQA and data cleaning
- CQA and schema matching/mapping

### Foundations
- defining measures of consistency
- more refined complexity analysis, dynamic aspects
Part III

XML
Outline of Part III

10 XML basics

11 XML keys and foreign keys

12 Consistency and implication

13 Applications
   - Integrity constraint propagation
   - XML normalization

14 Prospects

15 Valid Query Answers for XML
XML data model

- finite, ordered, unranked tree
- element, attribute and text nodes
Validity of XML documents

**XML data model**

- finite, ordered, unranked tree
- element, attribute and text nodes

**XML trees represent well-formed documents:**

- matching, properly nested opening and closing tags
- single root element
Validity of XML documents

**XML data model**
- finite, ordered, unranked tree
- element, attribute and text nodes

**XML trees represent** well-formed documents:
- matching, properly nested opening and closing tags
- single root element

**Valid XML documents**
- syntactic structure (**DTD**)
- syntactic structure and rich set of types (**XML Schema**)
- integrity constraints
<books>
  <book @title="1984">
    <author>G. Orwell</author>
    <part @num=1/>
    <part @num=2/>
    <citation @title="Utopia"/>
  </book>
  <book @title="Utopia">
  </book>
</books>
XML integrity constraints

What is familiar

- kinds of constraints: key, foreign key

What is new

- tree data model: nodes, paths
- different notions of equality: value-equality, node identity
- constraint scoping: absolute, relative, path-based
- interaction with syntax specifications
- no uniform framework
Document Type Definitions (DTDs)

**DTD**

- A finite set of **element types** $E$ (including the root type)
- A finite set of **attributes** $A$, where $A \cap E = \emptyset$

For each $\tau \in E$, the **content** $P(\tau)$ is a regular expression:

$$E ::= \varepsilon | \tau | E \cup E | E \ast$$

**Validity**

An XML tree is **valid** w.r.t. a DTD if for every node $n$ with label $\tau$ in the tree, the concatenation of the labels of the children of $\tau$ is in the regular language defined by $P(\tau)$.

**DTD: element types**
- books
- book*
- book
- author, part*, citation*

**DTD: attributes**
- book: @title
- citation: @title
- part: @num
Document Type Definitions (DTDs)

**DTD**

- a finite set of **element** types $E$ (incl. the **root** type)
- a finite set of **attributes** $A$ ($A \cap E = \emptyset$)
- for each $\tau \in E$, the **content** $P(\tau)$ is a regular expression:

$$E ::= \varepsilon \mid \tau' \mid E \cup E \mid E, E \mid E^*$$
Document Type Definitions (DTDs)

**DTD**
- a finite set of **element** types $E$ (incl. the **root** type)
- a finite set of **attributes** $A$ ($A \cap E = \emptyset$)
- for each $\tau \in E$, the **content** $P(\tau)$ is a regular expression:

$$E : = \varepsilon \mid \tau' \mid E \cup E \mid E, E \mid E^*$$

**Validity**
An XML tree is **valid** w.r.t. a DTD if for every node $n$ with label $\tau$ in the tree, the concatenation of the labels of the children of $\tau$ is in the regular language defined by $P(\tau)$. 
Document Type Definitions (DTDs)

**DTD**
- a finite set of element types $E$ (incl. the root type)
- a finite set of attributes $A$ ($A \cap E = \emptyset$)
- for each $\tau \in E$, the content $P(\tau)$ is a regular expression:
  \[
  E ::= \varepsilon | \tau' | E \cup E | E, E | E^*
  \]

**Validity**
An XML tree is *valid* w.r.t. a DTD if for every node $n$ with label $\tau$ in the tree, the concatenation of the labels of the children of $\tau$ is in the regular language defined by $P(\tau)$.

**DTD: element types**
- $books \leadsto book^*$
- $book \leadsto author, part^*, citation^*$
- $author \leadsto PCDATA$
- ...

**DTD: attributes**
- $book: @title$
- $citation: @title$
- $part: @num$
- ...
Keys and foreign keys (Buneman et al. [BDF+02])

Absolute vs. relative

- **absolute**: constraints hold over the entire document
- **relative**: constraints hold over subdocuments rooted at a given element type
Keys and foreign keys (Buneman et al. [BDF⁺02])

Absolute vs. relative

- **absolute**: constraints hold over the entire document
- **relative**: constraints hold over subdocuments rooted at a given element type

Absolute keys

A document satisfies a key \( \tau[X] \rightarrow \tau \) iff

\[
\forall u, v \in \text{ext}(\tau). \bigwedge_{A \in X} u.A = v.A \Rightarrow u = v
\]
Keys and foreign keys (Buneman et al. [BDF+02])

Absolute vs. relative
- **absolute**: constraints hold over the entire document
- **relative**: constraints hold over subdocuments rooted at a given element type

Absolute keys
A document satisfies a key $\tau[X] \rightarrow \tau$ iff
\[
\forall u, v \in \text{ext}(\tau). \bigwedge_{A \in X} u.A = v.A \Rightarrow u = v
\]

Notation
$\text{ext}(\tau)$: the set of $\tau$-element nodes in the document

Notions of equality
- **LHS**: string value equality
- **RHS**: node identity

Jan Chomicki ()
Database Consistency
June 25-29, 2007 68 / 85
Keys and foreign keys (Buneman et al. [BDF+02])

Absolute vs. relative

- **absolute**: constraints hold over the entire document
- **relative**: constraints hold over subdocuments rooted at a given element type

Absolute keys

A document satisfies a key $\tau[X] \rightarrow \tau$ iff

$$\forall u, v \in \text{ext}(\tau). \bigwedge_{A \in X} u.A = v.A \Rightarrow u = v$$

Notation

$\text{ext}(\tau)$: the set of $\tau$-element nodes in the document

Notions of equality

- **LHS**: string value equality
- **RHS**: node identity

Absolute foreign keys

A document satisfies a foreign key $(\tau_1[X] \subseteq \tau_2[Y], \tau_2[Y] \rightarrow \tau_2)$ iff

$$\forall u \in \text{ext}(\tau_1). \exists v \in \text{ext}(\tau_2). u[X] = v[Y]$$
Example XML document

```
books
  book
    title="1984"
    author="G. Orwell"
    part
      num=1
  part
    num=2
  citation
    title="Utopia"
```
Example XML document

```
books
  book
    @title="1984"
    author
      "G. Orwell"
    part
      @num=1
    part
      @num=2
  citation
    @title="Utopia"
  book
    @title="Utopia"
```

Integrity constraints

**Keys:**

\[ book.\texttt{@title} \rightarrow \textit{book} \]

\[ \textit{book}(\text{part.}\texttt{@num} \rightarrow \textit{part}) \]

**Foreign keys:**

\[ (\text{citation.}\texttt{@title} \subseteq \textit{book.}\texttt{@title}, \textit{book.}\texttt{@title} \rightarrow \textit{book}) \]
Path expressions

\[ E := \varepsilon \mid \tau' \mid E/E \mid E//E \]
Path constraints

Path expressions

\[ E := \varepsilon \mid \tau^' \mid E/E \mid E//E \]

Absolute key constraints

\((Q, \{P_1, \ldots, P_k\})::\)

- **Q**: target path to identify the target set of nodes \([Q]\) on which the key is defined
- **P_1, \ldots, P_k**: key paths to provide identification for the nodes in \([Q]\)
- **semantics**: for any two nodes in \([Q]\), if they have all the key paths and agree on them by value equality, then they must be the same node.
Path constraints

Path expressions

\[ E ::= \varepsilon | \tau' | E/E | E//E \]

Absolute key constraints

\((Q, \{P_1, \ldots, P_k\}):\)

- **Q:** target path to identify the target set of nodes \([Q]\) on which the key is defined
- **\(P_1, \ldots, P_k\):** key paths to provide identification for the nodes in \([Q]\)
- **semantics:** for any two nodes in \([Q]\), if they have all the key paths and agree on them by value equality, then they must be the same node.

Relative key constraints

\((Q_0, (Q, \{P_1, \ldots, P_k\})):\)

- **Q_0:** context path
- **(Q, \{P_1, \ldots, P_k\})** is a key on subdocuments rooted at the nodes in \([Q_0]\)
Path constraints

\[(\varepsilon, (//\text{book}, \{\@title\})))\]

\[(//\text{book}, (\text{part}, \{\@num\})))\]

\[(//\text{book}, (\text{author}, \emptyset))\]
(Absolute) key constraints

$(Q, \{P_1, \ldots, P_k\})$:

- $Q, P_1, \ldots, P_k$: (limited) XPath expression
- uniqueness and existence: for each node $x$ in $[Q]$ and each $i = 1, \ldots, k$, there is a single node $u_i$ (text or attribute) reached from $x$ via $P_i$
- identification: for different nodes in $[Q]$, at least one of paths in $P_1, \ldots, P_k$ results in different nodes.
(Absolute) key constraints

\((Q, \{P_1, \ldots, P_k\})\):

- \(Q, P_1, \ldots, P_k\): (limited) XPath expression
- **uniqueness and existence**: for each node \(x\) in \([Q]\) and each \(i = 1, \ldots, k\), there is a single node \(u_i\) (text or attribute) reached from \(x\) via \(P_i\)
- **identification**: for different nodes in \([Q]\), at least one of paths in \(P_1, \ldots, P_k\) results in different nodes.

(Absolute) foreign key constraints

\((Q, \{P_1, \ldots, P_k\}) \subseteq (S, \{T_1, \ldots, T_k\})\):

- key constraint \((S, \{T_1, \ldots, T_k\})\)
- **uniqueness and existence**: for both \(P_1, \ldots, P_k\) and \(T_1, \ldots, T_k\)
Consistency

Given a syntax specification $S$ and a set of integrity constraints $\Sigma$, is there a document valid w.r.t. $S$ and satisfying $\Sigma$?
Main problems

Consistency
Given a syntax specification $S$ and a set of integrity constraints $\Sigma$, is there a document valid w.r.t. $S$ and satisfying $\Sigma$?

Implication
Given a syntax specification $S$, a set of ICs $\Sigma$ and an IC $\sigma$, does every document valid w.r.t. $S$ and satisfying $\Sigma$ also satisfy $\sigma$?
Consistency is nontrivial

**DTD: element types**

<table>
<thead>
<tr>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>teachers</td>
<td>$\rightarrow$ teacher$^+$</td>
</tr>
<tr>
<td>teacher</td>
<td>$\rightarrow$ teach, research</td>
</tr>
<tr>
<td>teach</td>
<td>$\rightarrow$ subject, subject</td>
</tr>
<tr>
<td>subject</td>
<td>$\rightarrow$ PCDATA</td>
</tr>
<tr>
<td>research</td>
<td>$\rightarrow$ PCDATA</td>
</tr>
</tbody>
</table>

**DTD: attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>teacher</td>
<td>@name</td>
</tr>
<tr>
<td>subject</td>
<td>@by</td>
</tr>
</tbody>
</table>

**Integrity constraints**

- $teacher.@name \rightarrow teacher$
- $subject.@by \rightarrow subject$
- $subject.@by \subseteq teacher.@name$
Consistency is nontrivial

**DTD: element types**

- `teachers` ∼ `teacher^+`
- `teacher` ∼ `teach, research`
- `teach` ∼ `subject, subject`
- `subject` ∼ `PCDATA`
- `research` ∼ `PCDATA`

**DTD: attributes**

- `teacher`: `@name`
- `subject`: `@by`

**Integrity constraints**

- `teacher.@name` → `teacher`
- `subject.@by` → `subject`
- `subject.@by` ⊆ `teacher.@name`

**From the DTD**

\[
|\text{ext}(teacher)| < |\text{ext}(subject)|
\]
Consistency is nontrivial

**DTD: element types**

- teachers $\sim$ teacher$^+$
- teacher $\sim$ teach, research
- teach $\sim$ subject, subject
- subject $\sim$ PCDATA
- research $\sim$ PCDATA

**DTD: attributes**

- teacher: @name
- subject: @by

**Integrity constraints**

- teacher.@name $\rightarrow$ teacher
- subject.@by $\rightarrow$ subject
- subject.@by $\subseteq$ teacher.@name

**From the DTD**

- $|\text{ext}(\text{teacher})| < |\text{ext}(\text{subject})|$

**From the constraints**

- $|\text{ext}(\text{teacher.@name})| = |\text{ext}(\text{teacher})|$
- $|\text{ext}(\text{subject.@by})| = |\text{ext}(\text{subject})|$
- $|\text{ext}(\text{subject.@by})| \leq |\text{ext}(\text{teacher.@name})|$
- $\Rightarrow |\text{ext}(\text{subject})| \leq |\text{ext}(\text{teacher})|$
# Keys and foreign keys

<table>
<thead>
<tr>
<th></th>
<th>Absolute</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary</td>
<td>NP-complete</td>
<td>Undecidable</td>
</tr>
<tr>
<td>Multi-attribute</td>
<td>Undecidable</td>
<td>Undecidable</td>
</tr>
</tbody>
</table>
### Keys and foreign keys

<table>
<thead>
<tr>
<th></th>
<th>Absolute</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary</td>
<td>NP-complete</td>
<td>Undecidable</td>
</tr>
<tr>
<td>Multi-attribute</td>
<td>Undecidable</td>
<td>Undecidable</td>
</tr>
</tbody>
</table>

### Keys only

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-attribute relative</td>
<td>Linear time</td>
</tr>
<tr>
<td>XML Schema unary</td>
<td>NP-hard</td>
</tr>
</tbody>
</table>
### Keys and foreign keys

<table>
<thead>
<tr>
<th></th>
<th>Absolute</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary</td>
<td>NP-complete</td>
<td>Undecidable</td>
</tr>
<tr>
<td>Multi-attribute</td>
<td>Undecidable</td>
<td>Undecidable</td>
</tr>
</tbody>
</table>

### Keys only

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-attribute relative</td>
<td>Linear time</td>
</tr>
<tr>
<td>XML Schema unary</td>
<td>NP-hard</td>
</tr>
</tbody>
</table>

### Proof techniques

- multi-attribute constraints: reductions from relational problems
- unary constraints: polynomially equivalent to Linear Integer Programming
### Keys and foreign keys

<table>
<thead>
<tr>
<th></th>
<th>Absolute</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary</td>
<td>co-NP-complete</td>
<td>Undecidable</td>
</tr>
<tr>
<td>Multi-attribute</td>
<td>Undecidable</td>
<td>Undecidable</td>
</tr>
</tbody>
</table>

**Implication**

**Keys only**

- Multi-attribute absolute: Linear time
- XML Schema unary: co-NP-hard
- Simple relative path keys, no DTD: Quadratic time [HL07]
### Keys and foreign keys

<table>
<thead>
<tr>
<th>Absolute</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary</td>
<td>co-NP-complete</td>
</tr>
<tr>
<td>Multi-attribute</td>
<td>Undecidable</td>
</tr>
</tbody>
</table>

### Keys only

- Multi-attribute absolute: Linear time
- XML Schema unary: co-NP-hard
- Simple relative path keys, no DTD: Quadratic time [HL07]
XML shredding

- mapping XML documents to relations
- mapping XML keys to relation keys
**XML shredding**
- mapping XML documents to relations
- mapping XML keys to relation keys

**XML path keys**

- `//book, (chapter, {@num})`: chapter numbers unique within a book
- `//book, (title, {})`: each book has a single title ... which does not have to be
XML shredding

- mapping XML documents to relations
- mapping XML keys to relation keys

XML path keys

- `//@isbn)` globally unique ISBN
- `//book,(chapter,@num)` chapter numbers unique within a book
- `//book,(title,∅)` each book has a single title ... which does not have to be

Candidate relation?

`Chapter(Title, ChapterNum, ChapterTitle)`
Propagating relational constraints (Davidson, Fan, Hara[DFH07])

**XML shredding**

- mapping XML documents to relations
- mapping XML keys to relation keys

**XML path keys**

- $(//book, (chapter, \{@num\}))$ chapter numbers unique within a book
- $(//book, (title, \emptyset))$ each book has a single title ... which does not have to be

**Candidate relation?**

$Chapter(Title, ChapterNum, ChapterTitle)$

Will the key constraint of the relation $Chapter$ be propagated?
Correctness criterion

Assuming a set of XML keys $\Sigma$, a relation key $\alpha$ is propagated using a mapping $f$, if for every document $l$ satisfying $\Sigma$, the relation $f(l)$ satisfies $\alpha$. 
Which constraints are propagated?

Correctness criterion

Assuming a set of XML keys \( \Sigma \), a relation key \( \alpha \) is propagated using a mapping \( f \), if for every document \( I \) satisfying \( \Sigma \), the relation \( f(I) \) satisfies \( \alpha \).

Unsuccessful propagation

The key of \( \text{Chapter}(Title, ChapterNum, ChapterTitle) \) will not be propagated.
Which constraints are propagated?

**Correctness criterion**
Assuming a set of XML keys $\Sigma$, a relation key $\alpha$ is propagated using a mapping $f$, if for every document $I$ satisfying $\Sigma$, the relation $f(I)$ satisfies $\alpha$.

**Unsuccessful propagation**
The key of $Chapter(Title, ChapterNum, ChapterTitle)$ will not be propagated.

**Successful propagation**
A different schema: $Chapter(ISBN, ChapterNum, ChapterTitle)$. 
We need to adapt the notions of functional dependency, normal forms etc. to the context of XML.

**Tree tuple**

Assigns nodes, attribute values or nulls to paths:

- paths are **valid** w.r.t. a DTD
- paths are mapped to their last nodes in a consistent manner
XML normalization (Arenas, Libkin [AL04])

We need to adapt the notions of functional dependency, normal forms etc. to the context of XML.

**Tree tuple**

Assigns nodes, attribute values or nulls to paths:
- paths are valid w.r.t. a DTD
- paths are mapped to their last nodes in a consistent manner

**XFDs**

An XFD $\varphi = \{q_1, \ldots, q_n\} \rightarrow q$ is true in a document if for every tree tuples $t_1$ and $t_2$ of the document, whenever $t_1$ and $t_2$ agree on all $q_1, \ldots, q_n$ and are non-null, then they also agree on $q$. 
**Example**

**DTD: element types**

- $db \sim\rightarrow conf^*$
- $conf \sim\rightarrow issue^+$
- $issue \sim\rightarrow paper^+$

**DTD: attributes**

- $conf: @title$
- $paper: @title$
- $paper: @pages$
- $paper: @year$

**XFDs**

- $db.conf.@title \rightarrow db.conf$
- $db.conf.issue \rightarrow db.conf.issue.paper.@year$
Example

DTD: element types

- \( db \sim \text{conf}^* \)
- \( \text{conf} \sim \text{issue}^+ \)
- \( \text{issue} \sim \text{paper}^+ \)

DTD: attributes

- \( \text{conf}: @title \)
- \( \text{paper}: @title \)
- \( \text{paper}: @pages \)
- \( \text{paper}: @year \)

XFDs

- \( db.\text{conf}.@title \rightarrow db.\text{conf} \)
- \( db.\text{conf}.\text{issue} \rightarrow db.\text{conf}.\text{issue}.\text{paper}.@year \)

Are there any potential redundancies?
Normal form

Given a DTD $D$ and a set $\Sigma$ of XFDs, $(D, \Sigma)$ is in XNF if for every nontrivial XFD $X \rightarrow \rho$ implied by $(D, \Sigma)$, the XFD $X \rightarrow \rho$ is also implied by $(D, \Sigma)$.

Reaching XNF
The example document is not in XNF but can be transformed into XNF by moving the attribute year from paper to issue.

Computational complexity
The complexity of testing XFD implication ranges from quadratic time to co-NEXPTIME, depending on the form of the DTD.
Given a DTD $D$ and a set $\Sigma$ of XFDs, $(D, \Sigma)$ is in XNF if for every nontrivial XFD $X \rightarrow p.\forall A$ implied by $(D, \Sigma)$, the XFD $X \rightarrow p$ is also implied by $(D, \Sigma)$. 

Reaching XNF

The example document is not in XNF but can be transformed into XNF by moving the attribute `year` from `paper` to `issue`.

Computational complexity

The complexity of testing XFD implication ranges from quadratic time to co-NEXPTIME, depending on the form of the DTD.
**XNF**

Given a DTD $D$ and a set $\Sigma$ of XFDs, $(D, \Sigma)$ is in XNF if for every nontrivial XFD $X \rightarrow p.\circ A$ implied by $(D, \Sigma)$, the XFD $X \rightarrow p$ is also implied by $(D, \Sigma)$.

**Reaching XNF**

The example document is not in XNF but can be transformed into XNF by moving the attribute *year* from *paper* to *issue*. 
Given a DTD $D$ and a set $\Sigma$ of XFDs, $(D, \Sigma)$ is in XNF if for every nontrivial XFD $X \rightarrow p.\@A$ implied by $(D, \Sigma)$, the XFD $X \rightarrow p$ is also implied by $(D, \Sigma)$.

The example document is not in XNF but can be transformed into XNF by moving the attribute `year` from `paper` to `issue`.

The complexity of testing XFD implication ranges from quadratic time to co-NEXPTIME, depending on the form of the DTD.
XML constraints: the bottom line

The right language

- using path expressions to capture the scope and the contents of a constraint
- various proposals: no uniform syntax or semantics
- very preliminary logical formulations [DT05], equational chase
- applications: data shredding/publishing, schema mapping

Constraint analysis

- constraints and syntax specifications separately
- constraints and syntax specifications together: high complexity if both keys and foreign keys
Prospects for integrity constraints

**Semantic Web**
- knowledge bases and ontologies
- extensions of ICs
- relational representations

**Data mining**
- discovery of FDs and INDs

**Data cleaning**
M. Arenas, L. Bertossi, and J. Chomicki.
Consistent Query Answers in Inconsistent Databases.

M. Arenas, L. Bertossi, and J. Chomicki.
Answer Sets for Consistent Query Answering in Inconsistent Databases.

Consistency of XML Specifications.
In Bertossi et al. [BHS05], pages 15–41.

P. Andritsos, A. Fuxman, and R. Miller.
Clean Answers over Dirty Databases.
In IEEE International Conference on Data Engineering (ICDE), 2006.

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

M. Arenas and L. Libkin.
A Normal Form for XML Documents.

L. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko.
L. Bertossi and J. Chomicki.
Query Answering in Inconsistent Databases.

M. Baudinet, J. Chomicki, and P. Wolper.
Constraint-Generating Dependencies.
In International Conference on Database Theory (ICDT), pages 322–337, Prague, Czech Republic, January 1995. Springer-Verlag, LNCS 893.

Keys for XML.

L. Bertossi.
Consistent Query Answering in Databases.
SIGMOD Record, 35(2), June 2006.

A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification.

L. Bertossi, A. Hunter, and T. Schaub, editors.  
*Inconsistency Tolerance.*  
Springer-Verlag, 2005.

A. Borgida.  
Language Features for Flexible Handling of Exceptions in Information Systems.  

J. Chomicki.  
Preference Formulas in Relational Queries.  

J. Chomicki.  
Consistent Query Answering: Five Easy Pieces.  
Keynote talk.

J. Chomicki.  
Semantic optimization techniques for preference queries.  
A. Chandra, H.R. Lewis, and J.A. Makowsky.
Embedded Implicational Dependencies and their Inference Problem.

A. Calì, D. Lembo, and R. Rosati.
On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases.

A. Calì, D. Lembo, and R. Rosati.
Query Rewriting and Answering under Constraints in Data Integration Systems.

J. Chomicki and J. Marcinkowski.
Minimal-Change Integrity Maintenance Using Tuple Deletions.

J. Chomicki and J. Marcinkowski.
On the Computational Complexity of Minimal-Change Integrity Maintenance in Relational Databases.
In Bertossi et al. [BHS05], pages 119–150.

Computing Consistent Query Answers Using Conflict Hypergraphs.

A. Chandra and M. Vardi.

P. De Bra and J. Paredaens.
Conditional Dependencies for Horizontal Decompositions.
In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 123–141, 1983.

S. Davidson, W. Fan, and C. S. Hara.
Propagating XML constraints to relations.

W.F. Dowling and J. H. Gallier.
Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae.

A. Deutsch and V. Tannen.
XML Queries and Constraints, Containment and Reformulation.
T. Eiter, M. Fink, G. Greco, and D. Lembo.
Efficient Evaluation of Logic Programs for Querying Data Integration Systems.

Data Exchange: Semantics and Query Answering.

W. Fan and L. Libkin.
On XML Integrity Constraints in the Presence of DTDs.

A. Fuxman and R. J. Miller.
ConQuer: Efficient Management of Inconsistent Databases.

A. Fuxman and R. J. Miller.
First-Order Query Rewriting for Inconsistent Databases.
Full version to appear in JCSS.

G. Greco, S. Greco, and E. Zumpano.
A Logical Framework for Querying and Repairing Inconsistent Databases.
S. Hartmann and S. Link.
Unlocking Keys for XML Trees.

A. Lopatenko and L. Bertossi.
Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics.

J. Wijsen.
Database Repairing Using Updates.

J. Wijsen.