5. Data Exchange

- **Virtual Data Integration**
  - Never materialize instances for the global schema
  - Data of global schema only "visible" through queries
- **Data Exchange**
  - Materialize instance of global instance
  - We call it the "target schema"
  - Based on information from an instance of the local schema
  - We call this the "source schema"

5. Data Exchange

- **Data Exchange Problem Statement**
  - **Input:**
    - Given a source and a target schema
    - + instance of the source schema
    - + set of schema mappings (here st-tgds)
  - **Output:**
    - Instance of the target schema that fulfills constraints

Example: Types of Matching

\[ x, y, z, a : \text{Person}(x, y) \land \text{Address}(y, z, a) \rightarrow \exists b, c : \text{Person}(x, z, a, b, c) \]
5.1 Data Exchange Setting

**Definition: Data Exchange Setting**
Data Exchange setting is a tuple \((S, T, I, \Sigma)\)
- Schema \(S\)
- Schema \(T\)
- Instance \(I\)
- Mappings \(\Sigma\) from \(S\) to \(T\)

- Schema \(S'\)
- Schema \(T'\)
- Instance \(I\) of \(S'\)
- Mappings \(\Sigma\) from \(S\) to \(T'\)

5.1 Data Exchange Solutions

**Example: Solutions**

**Person**

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>1</td>
<td>(312) 123-4343</td>
</tr>
<tr>
<td>Alice</td>
<td>2</td>
<td>(312) 555-7777</td>
</tr>
<tr>
<td>Bob</td>
<td>3</td>
<td>(465) 123-1234</td>
</tr>
</tbody>
</table>

**Address**

<table>
<thead>
<tr>
<th>Street</th>
<th>City</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Ave</td>
<td>Chicago</td>
<td>(312) 555-7777</td>
</tr>
<tr>
<td>2nd St</td>
<td>New York</td>
<td>(465) 123-1234</td>
</tr>
</tbody>
</table>

\(x, y, z, a, b, c\):
\[ \text{Person}(x, y)^{\bot} \text{Address}(y, z, a)^{\bot} \rightarrow \exists b, c : \text{Person}(x, z, a, b, c) \]

Can we come up with a solution?

5.1 Number of Solutions

**How many solutions exists?**
- Depends on how whether we use existentially quantified variables in the mappings?
- i.e., do we have attributes for which we have to invent values?
- What attribute values do we allow?
  - Surely values from the source instance (active domain)
  - NULL?
    - Need multiple NULL values as placeholders for missing values that have to be the same
- Note that this is the open-world assumption
  - there are infinitely many solutions (if domains infinite)

**Target instance domain**
- Consider a universe \(U\)
  - Source instance can only use values from \(U\)
- Consider an infinite set \(N\) of labeled NULLs
  - Target instance can use these as placeholders for missing values
5.1 Data Exchange Solutions

Example: Multiple Solutions

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Office phone</th>
<th>Office address</th>
<th>Home phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pete</td>
<td>Chicago</td>
<td>(312) 501 0000</td>
<td>10100 S State</td>
<td></td>
</tr>
<tr>
<td>Alice</td>
<td>Chicago</td>
<td>(312) 605 7777</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>New York</td>
<td>(905) 101 1234</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.1 Certain answers (… again)

- Have multiple solutions
  - Define certain answers for queries as before
  - Every tuple t so that t is in the result of query Q over any valid solution J

- What’s new?
  - Want to materialize an instance so that computing certain answers over this instance is easy
    - Not immediately clear that this actually possible

5.1 Universal solutions

- Universal solution
  - Want a solution that is as general as possible
  - We call such most general solutions universal solutions
  - How do we know whether it is most general
    - We can map the tuples in this solution to any other less general solution by replacing unspecified values (labelled nulls) with actual data values

- Query answering with universal solutions
  - For UCQs: run query over universal instance
  - Remove tuples with labelled nulls
  - Result are the certain answers!

5.1 Data Exchange Solutions

Example: Solution generality

How general is solution (in terms of certain answers)?

Consider query
\[ Q(n) \rightarrow P(n,a,op,oa,hp) \]

For UCQs: run query over universal instance

5.1 Universal solutions

Definition: Homomorphism

A homomorphism \( h \) from instance \( J \) to instance \( J' \) maps the constants and nulls of \( J \) to the constants and nulls of \( J' \) and fulfills the following conditions:

- Constants are mapped onto themselves: \( h(c) = c \)
- Every tuple \( \langle a_1, ..., a_n \rangle \) in \( J \) is mapped to a tuple in \( J' \): 
  \[ h(a_1), ..., h(a_n) \] in \( J' \)

5.1 Universal solutions

Definition: Universal solution

Given data exchange setting (5.1.L), an instance \( J' \) is called an universal solution for a source instance \( J \) if it is a solution and for every other solution \( J' \) hold that:

- There exists a homomorphism from \( J \) to \( J' \)
5.1 Data Exchange Solutions

Example: Solution generality

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Office phone</th>
<th>Office address</th>
<th>Home phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>Chicago</td>
<td>(312) 123-4567</td>
<td>111 N. Plaza</td>
<td>345-6789</td>
</tr>
<tr>
<td>Alice</td>
<td>Chicago</td>
<td>(312) 123-4567</td>
<td>111 N. Plaza</td>
<td>345-6789</td>
</tr>
<tr>
<td>Bob</td>
<td>New York</td>
<td>(465) 123-4567</td>
<td>111 N. Plaza</td>
<td>345-6789</td>
</tr>
</tbody>
</table>

Above is universal solution

How to map to below non-universal solution?
Replace general labeled nulls with values:
X = Hometown, Y = 111X322X3454, C = other town,

Example: Solution generality

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Office phone</th>
<th>Office address</th>
<th>Home phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>Chicago</td>
<td>(312) 123-4567</td>
<td>111 N. Plaza</td>
<td>345-6789</td>
</tr>
<tr>
<td>Alice</td>
<td>Chicago</td>
<td>(312) 123-4567</td>
<td>111 N. Plaza</td>
<td>345-6789</td>
</tr>
<tr>
<td>Bob</td>
<td>New York</td>
<td>(465) 123-4567</td>
<td>111 N. Plaza</td>
<td>345-6789</td>
</tr>
</tbody>
</table>

5.2 Computing Solutions

• Can we use a database system to compute solutions?
  – Yes, systems such as Clio generate queries that compute universal solutions!
    • SQL
    • Java
    • XSLT (for XML docs)

• Generating Executable Transformations
  – How to preserve semantics of labeled nulls
    • n = n’ is true if we have the same labeled null only
    • n = n’ if one is a constant and the other one is a labeled null

5.2 Skolem Functions

• Skolem functions for labeled nulls
  – For each existential variable in a tgd we create a new skolem function
  – What should be the arguments of the function?
    • Naive: all universally quantified variables
    • Better: only relevant ones
5.2 Skolem Functions

Example: Skolem Functions

\[ \text{Person} \rightarrow \text{Name, Address, Age} \]

\[ \text{Person} \rightarrow \text{Name, Address, Office-address, Home-phone} \]

\[ \forall a, b, c, d, e : \text{Person}(a, b, c, d, e) \rightarrow \exists f, g \text{Person}(a, f, g) \land \text{Address}(f, b, c) \]

Introduce skolem functions sk1 and sk2 for f and g.

What arguments to choose for sk1 and sk2?

E.g., f should be fixed for a certain address and should not depend on the person.

5.2 Skolem Functions

Clio Schema Graph Algorithm

Nodes

– Create a graph with one node for every target attribute and one node for every target relation
– Also add nodes for source attribute if they are copied to the target according to the mapping

Edges

– Edges between a relation and its attributes
– Edges between target attributes that use the same variable
– Edges between source attributes and target attributes if they use the same variable
5.2 Skolem Functions

Example: Skolem Functions

\[ \text{Address} = \text{Office-phone} \]

Skolem functions

- Clio Schema Graph Algorithm
- Skolem functions
  - Derive skolem function arguments from the schema graph annotations of an element

Example: Skolem Functions

\[ \text{Person} \rightarrow 3f,g \text{Person}(a,f,g) \land \text{Address}(f,h,c) \]

For variable \( f \) (id, address) we assign \( sk_1(a,b,c) \)
For variable \( g \) (age) we assign \( sk_2(a,b,c) \)

5.2 Executable Transformations

Example: Skolem Functions

\[ \text{Person} \rightarrow 3f,g \text{Person}(a,f,g) \land \text{Address}(f,h,c) \]

For Person atom in RHS:

\[
\begin{align*}
\text{SELECT} & | \text{name} | \text{address} | \text{office-phone} \\
\text{WHERE} & | \text{name} | \text{address} | \text{office-phone} \\
\text{FROM} & | \text{Person} \\
\end{align*}
\]

For Address atom in RHS:

\[
\begin{align*}
\text{SELECT} & | \text{name} | \text{address} | \text{office-phone} \\
\text{WHERE} & | \text{name} | \text{address} | \text{office-phone} \\
\text{FROM} & | \text{Person} \\
\end{align*}
\]

5.3 Recap Data Exchange Steps

- Schema Matching
- Generate Schema Mappings
  - Use constraints
- Generate Executable Transformations
  - SQL, XSLT, XQuery
  - Skolems for missing value
- Run Transformations over source instance to generate target instance
  - Universal solution
5.3 Comparison with virtual integration

- Pay cost upfront instead of at query time
- Making decisions early vs. at query time
  - When generating a solution
  - Caution: bad decisions stick!
- **Universal solutions** allow efficient computation of certain types of queries using, e.g., SQL