Composing Schema Mappings: An Overview

Phokion G. Kolaitis

UC Santa Scruz & IBM Almaden

Joint work with
Ronald Fagin, Lucian Popa, and Wang-Chiew Tan
The Data Interoperability Challenge

- Data may reside
  - at several different sites
  - in several different formats (relational, XML, ...).

- Applications need to access and process all these data.

- Growing market of enterprise data interoperability tools:
  - $1.44B in 2007; 17% annual rate of growth
  - 15 major vendors in Gartner’s Magic Quadrant Report
    (source: Gartner, Inc., September 2008)
Gartner’s Magic Quadrant Report on Data Interoperability Products

<table>
<thead>
<tr>
<th>Ability to execute</th>
<th>Challengers</th>
<th>Leaders</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Microsoft</td>
<td>Informatica</td>
</tr>
<tr>
<td></td>
<td>Oracle</td>
<td>IBM (Cognos)</td>
</tr>
<tr>
<td></td>
<td>Synco</td>
<td>SAP – Business Objects</td>
</tr>
<tr>
<td></td>
<td>ETI</td>
<td>SAS</td>
</tr>
<tr>
<td></td>
<td>Pitney Boss Software</td>
<td>Pervasive Software</td>
</tr>
<tr>
<td></td>
<td>Open Text</td>
<td>iWay Software</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sun Microsystems</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tibco Software</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Niche Players</th>
<th>Visionaries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness of vision</td>
<td></td>
</tr>
</tbody>
</table>
The research community has studied two different, but closely related, facets of data interoperability:

- **Data Integration** (aka Data Federation)
  - Formalized and studied for the past 10-15 years

- **Data Exchange** (aka Data Translation)
  - Formalized and studied for the past 5-6 years
Data Integration

Query heterogeneous data in different sources via a virtual global schema
Data Exchange

Transform data structured under a source schema into data structured under a different target schema.
Schema Mappings

- **Schema mappings:**
  High-level, declarative assertions that specify the relationship between two database schemas.

- Schema mappings constitute the essential **building blocks** in formalizing and studying data interoperability tasks, including **data integration** and **data exchange**.

- Schema mappings make it possible to separate the **design** of the relationship between schemas from its **implementation**.
  - Are easier to generate and manage (semi)-automatically;
  - Can be compiled into SQL/XSLT scripts automatically.
Schema Mappings

- **Schema Mapping** $M = (S, T, \Sigma)$
  - *Source* schema $S$, *Target* schema $T$
  - High-level, declarative assertions $\Sigma$ that specify the relationship between $S$ and $T$.

**Question:** What is a “good” schema-mapping specification language?
Schema-Mapping Specification Languages

- **Obvious Idea:**
  Use a logic-based language to specify schema mappings. In particular, use first-order logic.

- **Warning:**
  Unrestricted use of first-order logic as a schema-mapping specification language gives rise to **undecidability** of basic algorithmic problems about schema mappings.
Let us consider some simple tasks that every schema-mapping specification language should support:

- **Copy (Nicknaming):**
  - Copy each source table to a target table and rename it.

- **Projection:**
  - Form a target table by projecting on one or more columns of a source table.

- **Column Augmentation:**
  - Form a target table by adding one or more columns to a source table.

- **Decomposition:**
  - Decompose a source table into two or more target tables.

- **Join:**
  - Form a target table by joining two or more source tables.

- **Combinations of the above** (e.g., “join + column augmentation + ...”)

Schema Mapping Specification Languages

- **Copy (Nicknaming):**
  \[ \forall x_1, \ldots, x_n (P(x_1, \ldots, x_n) \rightarrow R(x_1, \ldots, x_n)) \]
- **Projection:**
  \[ \forall x, y, z (P(x, y, z) \rightarrow R(x, y)) \]
- **Column Augmentation:**
  \[ \forall x, y (P(x, y) \rightarrow \exists z R(x, y, z)) \]
- **Decomposition:**
  \[ \forall x, y, z (P(x, y, z) \rightarrow R(x, y) \land T(y, z)) \]
- **Join:**
  \[ \forall x, y, z (E(x, z) \land F(z, y) \rightarrow R(x, y, z)) \]
- **Combinations of the above** (e.g., “join + column augmentation + ...”)
  \[ \forall x, y, z (E(x, z) \land F(z, y) \rightarrow \exists w (R(x, y) \land T(x, y, z, w))) \]
Schema Mapping Specification Languages

- **Question:** What do all these tasks (copy, projection, column augmentation, decomposition, join) have in common?

- **Answer:**
  - They can be specified using tuple-generating dependencies (tgds).
  - In fact, they can be specified using a special class of tuple-generating dependencies known as source-to-target tuple generating dependencies (s-t tgds).
Database Integrity Constraints

- **Dependency Theory**: extensive study of integrity constraints in relational databases in the 1970s and 1980s (Codd, Fagin, Beeri, Vardi ...)

- **Tuple-generating dependencies** (tgds) emerged as an important class of constraints with a balance between high expressive power and good algorithmic properties. Tgds are expressions of the form

  \[ \forall x \ (\varphi(x) \rightarrow \exists y \ \psi(x, y)) \]

  where 

  \[ \varphi(x), \ \psi(x, y) \] are conjunctions of atomic formulas.

**Special Cases:**

- Inclusion Dependencies
- Multivalued Dependencies
The relationship between source and target is given by source-to-target tuple generating dependencies (s-t tgds)

\[ \forall x (\varphi(x) \rightarrow \exists y \psi(x, y)), \text{ where} \]

- \( \varphi(x) \) is a conjunction of atoms over the source;
- \( \psi(x, y) \) is a conjunction of atoms over the target.

s-t tgds assert that: some conjunctive query over the source is contained in some other conjunctive query over the target.

**Example:** (dropping the universal quantifiers in the front)

\((\text{Student } (s) \land \text{Enrolls}(s,c)) \rightarrow \exists t \exists g (\text{Teaches}(t,c) \land \text{Grade}(s,c,g))\)
Schema Mapping Specification Language

Fact: s-t tgds generalize the main specifications used in data integration:

- They generalize LAV (local-as-view) specifications:
  \[ P(x) \rightarrow \exists y \, \psi(x, y), \]  
  where \( P \) is a source schema.
  
- \( E(x, y) \rightarrow \exists z \, (H(x, z) \land H(z, y)) \) (LAV constraint)

Note: Copy, projection, and decomposition are LAV s-t tgds.

- They generalize GAV (global-as-view) specifications:
  \[ \varphi(x) \rightarrow R(x), \]  
  where \( R \) is a target relation
  
  (they are equivalent to full tgds: \( \varphi(x) \rightarrow \psi(x), \) 
  where \( \varphi(x) \) and \( \psi(x) \) are conjunctions of atoms).

- \( E(x, y) \land E(y, z) \rightarrow F(x, z) \) (GAV (full) constraint)

Note: Copy, projection, and join are GAV s-t tgds.
Data Exchange via the schema mapping $M = (S, T, \Sigma)$

Given a source instance $I$, construct a target instance $J$, so that $(I, J)$ satisfy the specifications $\Sigma$ of $M$.

Such a $J$ is called a solution for $I$.

Difficulty:
- Usually, there are multiple solutions
- Which one is the “best” to materialize?
Data Exchange & Universal solutions

Fagin, K ..., Miller, Popa:
Identified and studied the concept of a universal solution for schema mappings specified by s-t tgd's

- A universal solution is a most general solution.
- A universal solution "represents" the entire space of solutions.
- A "canonical" universal solution can be generated efficiently using the chase procedure.
- A universal solution can be used to compute the certain answers of conjunctive queries over the target schema.
Managing Schema Mappings

- Schema mappings can be quite complex.

- Methods and tools are needed to automate or semi-automate schema-mapping management.

- Metadata Management Framework – Bernstein 2003 based on generic schema-mapping operators:
  - Match operator
  - Merge operator
  - ...
  - Composition operator
  - Inverse operator
Composing Schema Mappings

- Given $M_{12} = (S_1, S_2, \Sigma_{12})$ and $M_{23} = (S_2, S_3, \Sigma_{23})$, derive a schema mapping $M_{13} = (S_1, S_3, \Sigma_{13})$ that is “equivalent” to the sequential application of $M_{12}$ and $M_{23}$.

- $M_{13}$ is a composition of $M_{12}$ and $M_{23}$

$$M_{13} = M_{12} \circ M_{23}$$
Composing Schema Mappings

Given $M_{12} = (S_1, S_2, \Sigma_{12})$ and $M_{23} = (S_2, S_3, \Sigma_{23})$, derive a schema mapping $M_{13} = (S_1, S_3, \Sigma_{13})$ that is “equivalent” to the sequence $M_{12}$ and $M_{23}$.

What does it mean for $M_{13}$ to be “equivalent” to the composition of $M_{12}$ and $M_{23}$?
Earlier Work

- **Metadata Model Management** (Bernstein in CIDR 2003)
  - Composition is one of the fundamental operators
  - However, no precise semantics is given

- **Composing Mappings among Data Sources** (Madhavan & Halevy in VLDB 2003)
  - First to propose a semantics for composition
  - However, their definition is in terms of maintaining the same certain answers relative to a class of queries.
  - Their notion of composition *depends* on the class of queries; it may *not* be unique up to logical equivalence.
Semantics of Composition

- Every schema mapping $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ defines a binary relationship $\text{Inst}(\mathbf{M})$ between instances:
  \[
  \text{Inst}(\mathbf{M}) = \{ (I,J) \mid (I,J) \models \Sigma \}.
  \]
  In fact, from a semantic point of view, a schema mapping $\mathbf{M}$ can be identified with the set $\text{Inst}(\mathbf{M})$.

- **Definition:** (FKPT)
  A schema mapping $\mathbf{M}_{13}$ is a composition of $\mathbf{M}_{12}$ and $\mathbf{M}_{23}$ if
  \[
  \text{Inst}(\mathbf{M}_{13}) = \text{Inst}(\mathbf{M}_{12}) \circ \text{Inst}(\mathbf{M}_{23}),
  \]
  that is,
  \[
  (I_1,I_3) \models \Sigma_{13}
  \]
  if and only if
  there exists $I_2$ such that $(I_1,I_2) \models \Sigma_{12}$ and $(I_2,I_3) \models \Sigma_{23}$. 

The Composition of Schema Mappings

**Fact:** If both $M = (S_1, S_3, \Sigma)$ and $M' = (S_1, S_3, \Sigma')$ are compositions of $M_{12}$ and $M_{23}$, then $\Sigma$ are $\Sigma'$ are logically equivalent. For this reason:

- We say that $M$ (or $M'$) is *the composition* of $M_{12}$ and $M_{23}$.
- We write $M_{12} \circ M_{23}$ to denote it.
Issues in Composition of Schema Mappings

- The **semantics** of composition was the first main issue.

- The second main issue is the **language** of the composition.

  - Is the language of s-t tgds *closed under composition*? If \( M_{12} \) and \( M_{23} \) are specified by finite sets of s-t tgds, is \( M_{12} \circ M_{23} \) also specified by a finite set of s-t tgds?

  - If not, what is the “right” language for composing schema mappings?
Inexpressibility of Composition

**Theorem:**
- The language of s-t tgdgs is **not** closed under composition.
- In fact, there are schema mappings $M_{12}$ and $M_{23}$ specified by s-t tgdgs such that their composition $M_{12} \circ M_{23}$ is **not** expressible in least fixed-point logic LFP; hence, it is expressible neither in first-order logic nor in Datalog.
Lower Bounds for Composition

- $M_{12}$:
  \[ \forall x \forall y \ (E(x,y) \rightarrow \exists u \exists v \ (C(x,u) \land C(y,v))) \]
  \[ \forall x \forall y \ (E(x,y) \rightarrow F(x,y)) \]

- $M_{23}$:
  \[ \forall x \forall y \forall u \forall v \ (C(x,u) \land C(y,v) \land F(x,y) \rightarrow D(u,v)) \]

Given graph $G=(V, E)$:
- Let $I_1 = E$
- Let $I_3 = \{ (r,g), (g,r), (b,r), (r,b), (g,b), (b,g) \}$

**Fact:**
$G$ is 3-colorable iff $<I_1, I_3> \in \text{Inst}(M_{12}) \circ \text{Inst}(M_{23})$

**Theorem (Dawar – 1998):**
3-Colorability is **not** expressible in LFP
**Complexity of Composition**

**Definition:** The model checking problem for a schema mapping \( M = (S, T, \Sigma) \) asks: given a source instanced I and a target instance J, does \(<I,J> \models \Sigma? \)

**Fact:** If a schema mapping M is specified by s-t tgds, then the model checking problem for M is in LOGSPACE.

**Fact:** There are schema mappings \( M_{12} \) and \( M_{23} \) specified by s-t tgds such that the model checking problem for their composition \( M_{12} \circ M_{23} \) is NP-complete.
Employee Example

- $\Sigma_{12}$:
  - $\text{Emp}(e) \rightarrow \exists m \text{ Rep}(e,m)$

- $\Sigma_{23}$:
  - $\text{Rep}(e,m) \rightarrow \text{Mgr}(e,m)$
  - $\text{Rep}(e,e) \rightarrow \text{SelfMgr}(e)$

**Theorem:** This composition is not definable by any set (finite or infinite) of s-t tgds.

**Fact:** This composition is definable in a well-behaved fragment of second-order logic, called SO tgds, that extends s-t tgds with Skolem functions.
Employee Example - revisited

\[ \Sigma_{12} : \]
- \( \forall e \ ( \text{Emp}(e) \rightarrow \exists m \ \text{Rep}(e,m) ) \)

\[ \Sigma_{23} : \]
- \( \forall e \forall m( \text{Rep}(e,m) \rightarrow \text{Mgr}(e,m) ) \)
- \( \forall e \ ( \text{Rep}(e,e) \rightarrow \text{SelfMgr}(e) ) \)

**Fact:** The composition is definable by the SO-tgd

\[ \Sigma_{13} : \]
- \( \exists f \ ( \forall e \ ( \text{Emp}(e) \rightarrow \text{Mgr}(e,f(e)) ) \land \forall e( \text{Emp}(e) \land (e=f(e)) \rightarrow \text{SelfMgr}(e) ) ) \)
Second-Order Tgds

**Definition:** Let $S$ be a source schema and $T$ a target schema. A second-order tuple-generating dependency (SO tgd) is a formula of the form:

$$
\exists f_1 \ldots \exists f_m ( (\forall x_1(\phi_1 \rightarrow \psi_1)) \land \ldots \land (\forall x_n(\phi_n \rightarrow \psi_n)) ),
$$

- Each $f_i$ is a function symbol.
- Each $\phi_i$ is a conjunction of atoms from $S$ and equalities of terms.
- Each $\psi_i$ is a conjunction of atoms from $T$.

**Example:**

$$
\exists f (\forall e(\text{Emp}(e) \rightarrow \text{Mgr}(e, f(e))) \land \\
\forall e(\text{Emp}(e) \land (e=f(e)) \rightarrow \text{SelfMgr}(e)))
$$
Composing SO-Tgds and Data Exchange

**Theorem** (FKPT):
- The composition of two SO-tgds is definable by a SO-tgd.
- There is an algorithm for composing SO-tgds.
- The chase procedure can be extended to SO-tgds; it produces universal solutions in polynomial time.
- Every SO tgd is the composition of finitely many finite sets of s-t tgds. Hence, SO tgds are the “right” language for the composition of s-t tgds.
When is the composition FO-definable?

**Fact:**
- It is an undecidable problem to tell whether the composition of two schema mappings specified by s-t tgds is first-order definable.
- However, there are certain sufficient conditions that guarantee that the composition is first-order definable.
  - If $M_{12}$ is specified by GAV (full) s-t tgds and $M_{23}$ is specified by s-t tgds, then their composition is definable by s-t tgds.
  - **Arocena, Fuxman, Miller:** If both $M_{12}$ and $M_{23}$ are specified by LAV s-t tgds with distinct variables, then their composition is specified by LAV s-t tgds.
Synopsis of Schema Mapping Composition

- s-t tgds are *not* closed under composition.
- SO-tgds form a *well-behaved* fragment of second-order logic.
  - SO-tgds are closed under composition; they are the “right” language for composing s-t tgds.
  - SO-tgds are “chasable”:
    Polynomial-time data exchange with universal solutions.
- SO-tgds and the composition algorithm have been incorporated in Clio’s *Mapping Specification Language* (MSL).
Related Work and Open Problems

**Related Work:**
- Composing richer schema mappings
  Nash, Bernstein, Melnik – 2007
- Composing Schema Mappings in Open & Closed Worlds
  Libkin and Sirangelo – 2008
- XML Schema Mappings
  Amano, Libkin, Murlak – 2009

**Open Problems:**
- Composition of schema mappings specified by s-t tgds and target constraints (target tgds and target egds).
- Composition of schema mappings specified by richer source-to-target dependencies.
“The notion of composition of maps leads to the most natural account of fundamental notions of mathematics, from multiplication, addition, and exponentiation, through the basic notions of logic.”

"Conceptual Mathematics"
by
Lawevere and Schanuel