Composing and Inverting

Schema Mappings

Phokion G. Kolaitis

University of California Santa Cruz
&
IBM Research - Almaden

The Information Integration 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 information integration tools:
  - About $1.5B per year; 17% annual rate of growth.
  - Information integration consumes 40% of the budget of enterprise information technology shops.

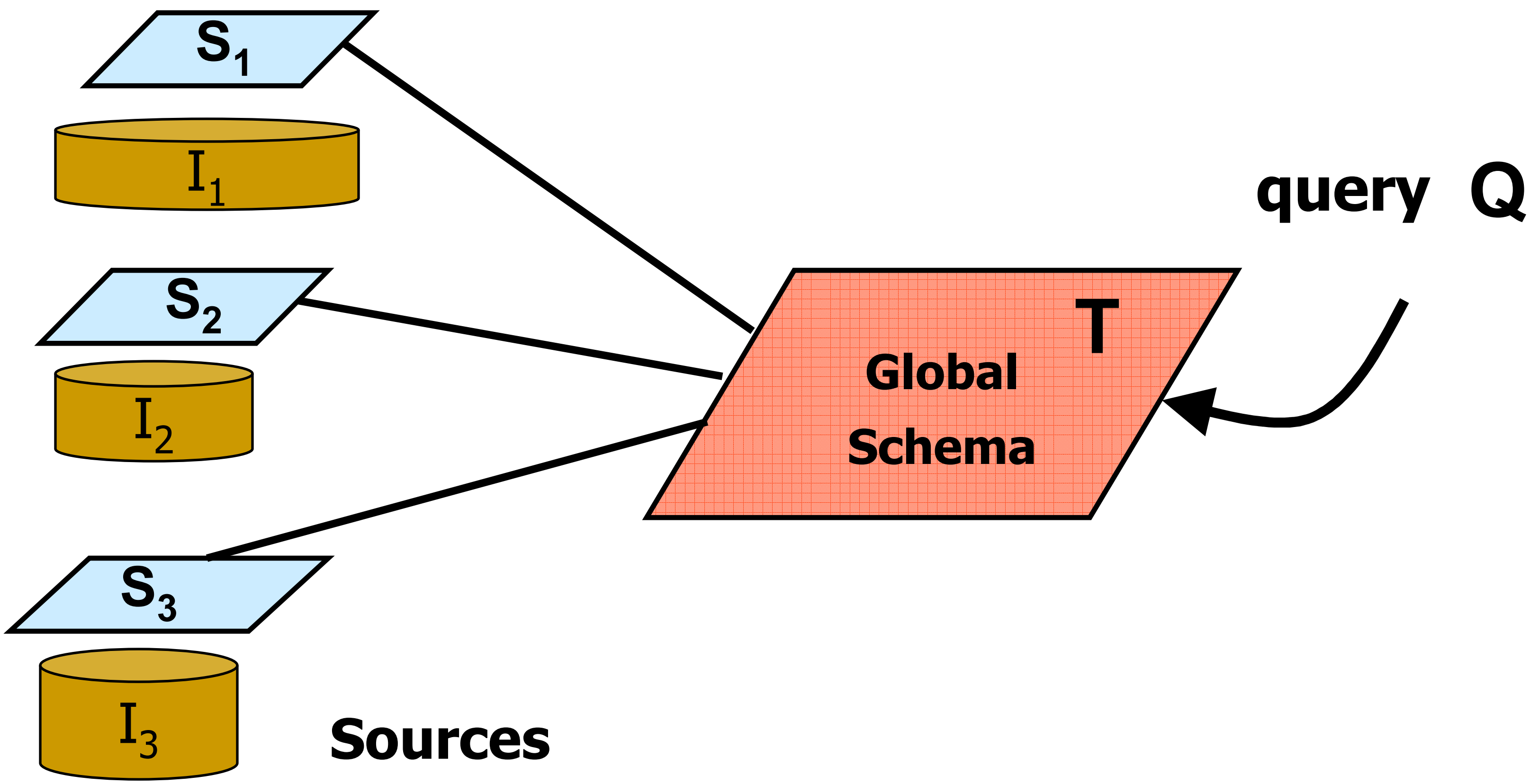
Two Facets of Information Integration

The research community has studied two different, but closely related, facets of information integration:

- **Data Integration** (aka **Data Federation**)
- **Data Exchange** (aka **Data Translation**)

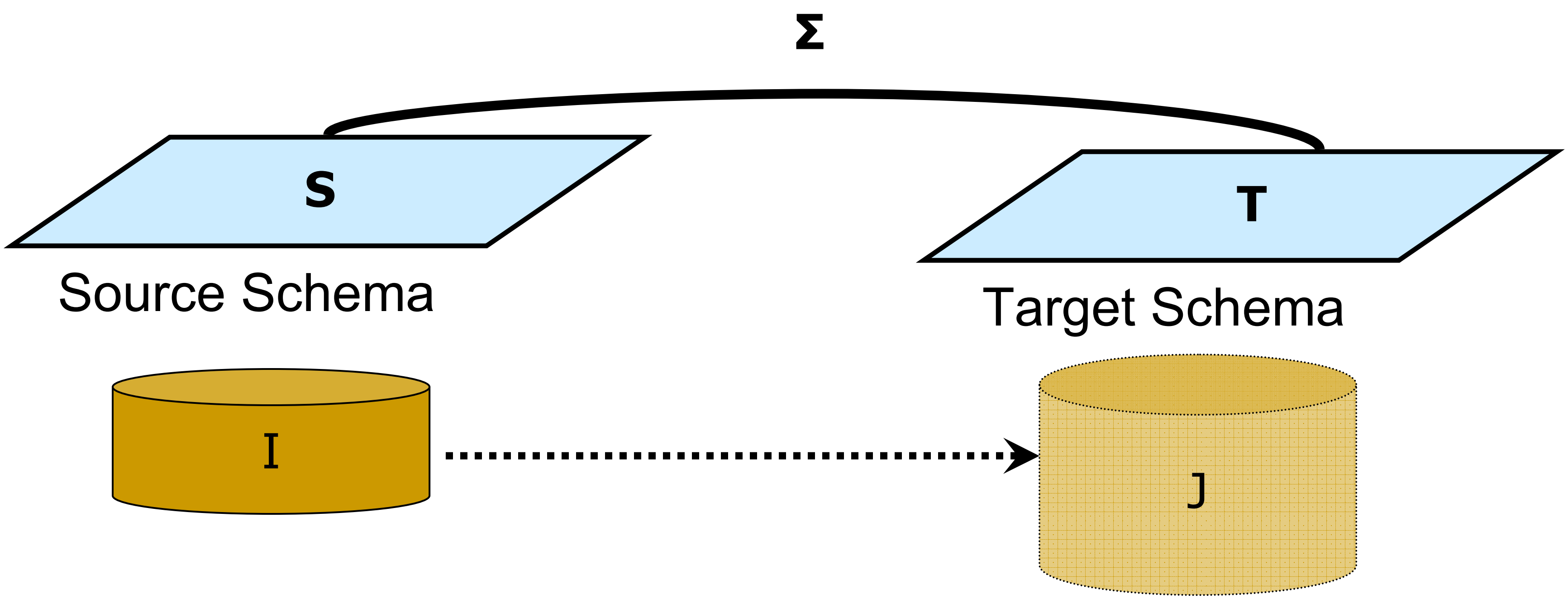
Data Integration

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 constitute the essential building blocks in formalizing and studying data integration and data exchange.

- Schema mappings are:

  High-level, declarative assertions that specify the relationship between two database schemas.

- 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.

Outline

- Schema Mappings
  - Schema-Mapping Specification Languages
  - Data Exchange: Semantics and the Chase Procedure
- Composing Schema Mappings
- Inverting Schema Mappings

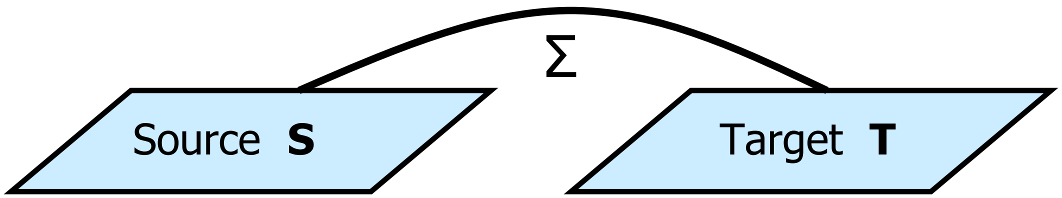
Acknowledgments

- Much of the work presented has been carried out in collaboration with
  - Ron Fagin, IBM Almaden
  - Renee J. Miller, University of Toronto
  - Lucian Popa, IBM Almaden
  - Wang-Chiew Tan, UC Santa Cruz.

  Papers in ICDT 2003, PODS 2003-2010, TCS, ACM TODS.

- The work has been motivated from the Clio Project at IBM Almaden aiming to develop a working system for schema-mapping generation and data exchange.

Schema Mappings



- Schema Mapping **M** = (**S**, **T**, Σ)
  - Source schema **S**, Target schema **T**
  - High-level, declarative assertions Σ 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.

Schema-Mapping Specification Languages

Every schema-mapping specification language should support:

- Copy (Nicknaming):
  - Copy each source table to a target table and rename it.
- Projection (Column Deletion):
  - Form a target table by deleting one or more columns of a source table.
- Column Addition:
  - 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 addition+ …")

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 Addition:
  - $\forall x,y\ (P(x,y) \rightarrow \exists\, z\ R(x,y,z))$
- Decomposition:
  - $\forall x,y,z\ (P(x,y,z) \rightarrow R(x,y) \wedge T(y,z))$
- Join:
  - $\forall x,y,z(E(x,z) \wedge F(z,y) \rightarrow R(x,z,y))$
- Combinations of the above (e.g., "join + column addition + …"):
  - $\forall x,y,z(E(x,z) \wedge F(z,y) \rightarrow \exists\, w\ (R(x,y) \wedge T(x,y,z,w)))$

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).

Schema-Mapping Specification Language

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

$$\forall \mathbf{x}\ (\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})),\ \text{where}$$

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

Examples:

- $\forall s\ \forall c$ (Student (s) $\wedge$ Enrolls(s,c) $\rightarrow$ $\exists g$ Grade(s,c,g))
- (dropping the universal quantifiers in the front)

Student (s) $\wedge$ Enrolls(s,c) $\rightarrow$ $\exists t\ \exists g$ (Teaches(t,c) $\wedge$ Grade(s,c,g))

Schema-Mapping Specification Language

Fact: s-t tgds are also known as

GLAV (global-and-local-as-view) constraints:

- They generalize **LAV (local-as-view)** constraints:

  $\forall \mathbf{x} \, (P(\mathbf{x}) \rightarrow \exists \mathbf{y} \, \psi(\mathbf{x}, \mathbf{y}))$, where P is a source relation.

- They generalize **GAV (global-as-view)** constraints:

  $\forall \mathbf{x} \, (\varphi(\mathbf{x}) \rightarrow R(\mathbf{x}))$, where R is a target relation.

LAV and GAV Constraints

Examples of LAV (local-as-view) constraints:

- Copy and projection
- Decomposition: $\forall x\ \forall y\ \forall z\ (P(x,y,z) \rightarrow R(x,y) \wedge T(y,z))$
- $\forall x\ \forall y\ (E(x,y) \rightarrow \exists\ z\ (H(x,z) \wedge H(z,y)))$

Examples of GAV (global-as-view) constraints:

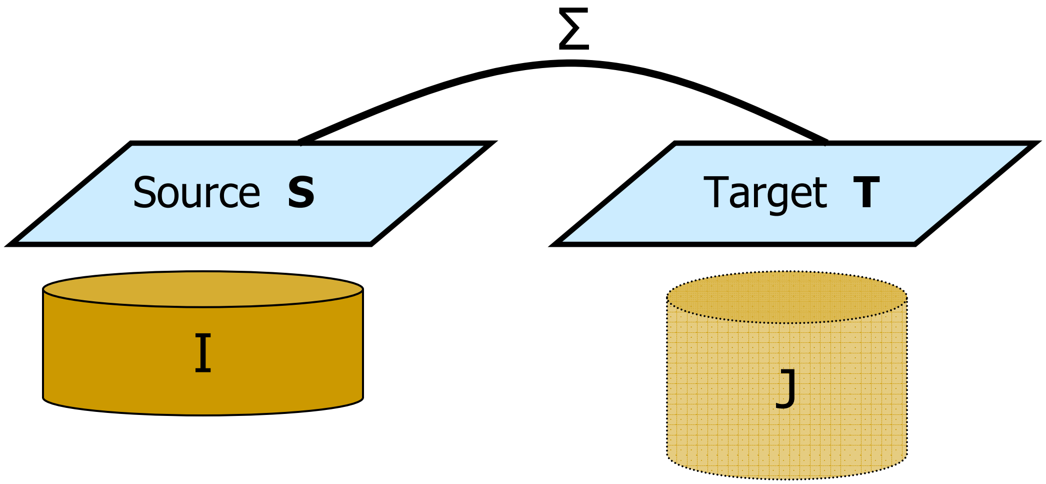
- Copy and projection
- Join: $\forall x\ \forall y\ \forall z\ (E(x,y) \wedge E(y,z) \rightarrow F(x,z))$

Note:

$$\forall s\ \forall c\ (Student\ (s) \wedge Enrolls(s,c) \rightarrow \exists g\ Grade(s,c,g))$$

is a GLAV constraint that is neither a LAV nor a GAV constraint

Semantics of Schema Mappings

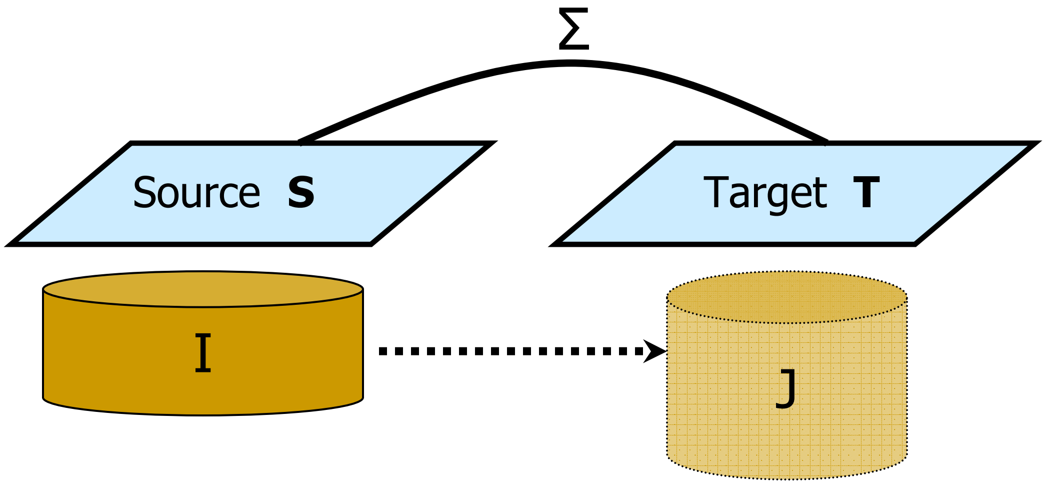


- **M** = (**S**, **T**, Σ) schema mapping with Σ a set of s-t tgds
- From a semantic point of view, **M** can be identified with

  Inst(**M**) = { (I,J): I is a sourse instance,
  J is a target instance, and (I,J) ⊨ Σ }

  (this is OWA semantics)
- A **solution** for a source instance I is a target instance J such that (I,J) ∈ Inst(**M**) (i.e., (I,J) ⊨ Σ).

Schema Mappings & Data Exchange



- **Data Exchange** via the schema mapping **M** = (**S**, **T**, Σ): Given a source instance I, construct a solution J for I.

- Difficulty:
  - Typically, 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** in data exchange.

- 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**.

Universal Solutions in Data Exchange

Note: Two types of values in instances:

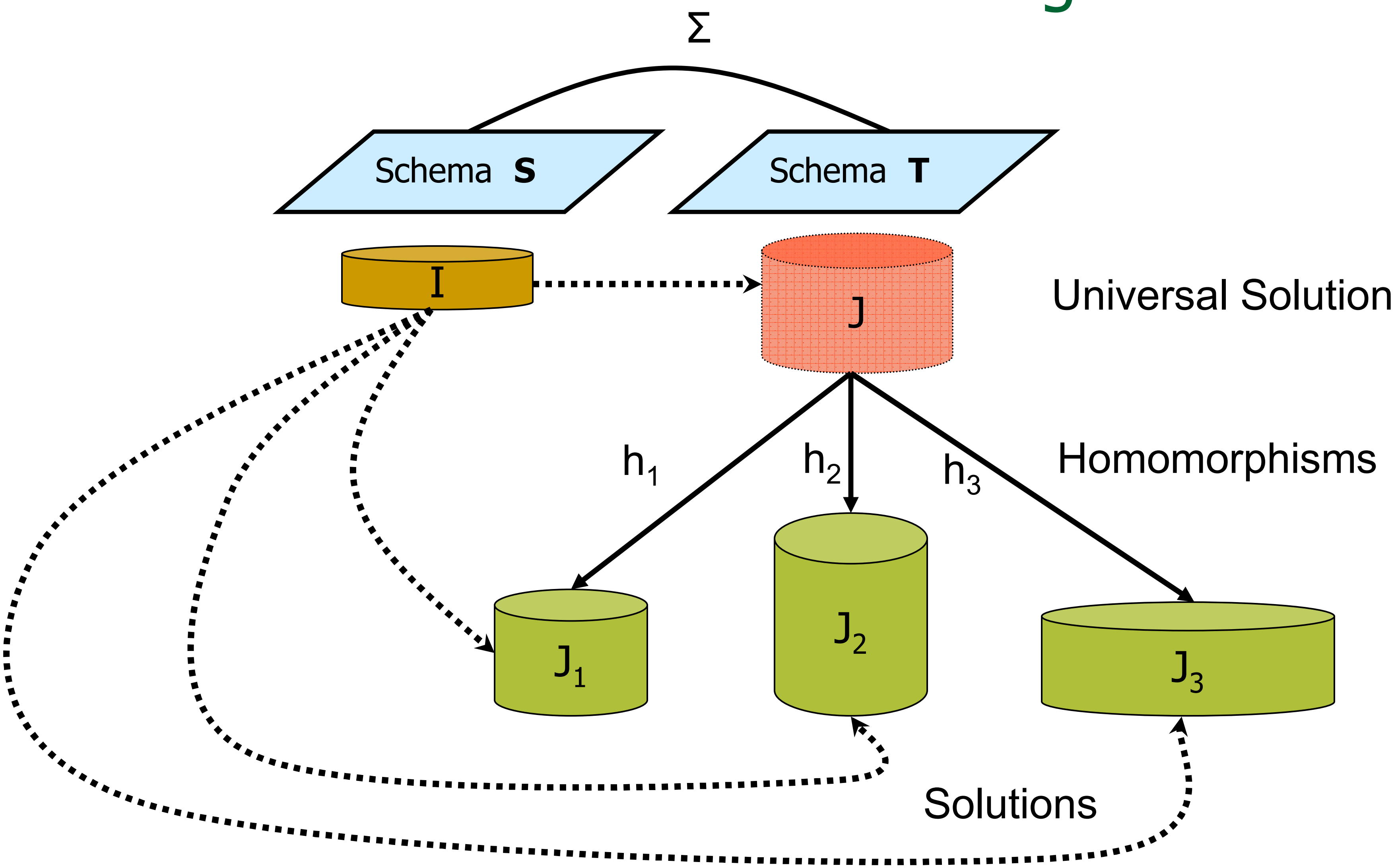
- **Constants**: they can only be mapped to themselves
- **Variables** (labeled nulls): they can be mapped to other values

Definition: Homomorphism h: J → K between instances:

- h(c) = c, for constant c
- If $P(a_1,\ldots,a_m)$ is in J, then $P(h(a_1),\ldots,h(a_m))$ is in K.

Definition (FKMP): A solution J for I is **universal** if it has homomorphisms to all other solutions for I.
(thus, a universal solution is a "most general" solution).

Universal Solutions in Data Exchange



Example

Source relation E(A,B), target relation F(A,B)

Σ : $E(x,y) \rightarrow \exists z\, (H(x,z) \wedge H(z,y))$

Source instance I = { E(1,2) }, where 1 and 2 are constants.

Solutions: Infinitely many solutions exist

- J_1 = { H(1,2), H(2,2) } is not universal
- J_2 = { H(1,1), H(1,2) } is not universal
- J_3 = { H(1,X), H(X,2) } is universal
- J_4 = { H(1,X), H(X,2), H(1,Y), H(Y,2) } is universal
- J_5 = { H(1,X), H(X,2), H(Y,Y) } is not universal

The Chase Procedure

Chase Procedure for **M** = (**S**, **T**, Σ) : given a source instance I, build a target instance $chase_{\mathbf{M}}(I)$ that satisfies every s-t tgd in Σ as follows.

Whenever the LHS of some s-t tgd in Σ evaluates to true:

- Introduce new facts in $chase_{\mathbf{M}}(I)$ as dictated by the RHS of the s-t tgd.
- In these facts, each time existential quantifiers need witnesses, introduce new variables (labeled nulls) as values.

The Chase Procedure

Example: Transforming edges to paths of length 2
M = (**S**, **T**, Σ) LAV schema mapping with
Σ : $E(x,y) \rightarrow \exists z(F(x,z) \wedge F(z,y))$

The chase returns a relation obtained from E by adding a new node between every edge of E.

- If I = { E(1,2) }, then $chase_{\mathbf{M}}(I)$ ={ E(1,X), E(X,2) }
- If I = { E(1,2), E(2,3), E(1,4) }, then
  $chase_{\mathbf{M}}(I)$ = { E(1,X), E(X,2), E(2,Y), E(Y,3), E(1,Z), E(Z,4) }

The Chase Procedure

Example : Collapsing paths of length 2 to edges

$\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ GAV schema mapping with

Σ : $E(x,z) \wedge E(z,y) \rightarrow F(x,y)$

- If $I = \{ E(1,3), E(2,4), E(3,4) \}$, then $chase_{\mathbf{M}}(I) = \{ F(1,4) \}$.
- If $I = \{ E(1,3), E(2,4), E(3,4), E(4,3)\}$, then $chase_{\mathbf{M}}(I) = \{ F(1,4), F(2,3), F(3,3), F(4,4) \}$.

Note: **No** new variables are introduced in the GAV case.

The Chase Procedure

Theorem (FKMP): Let **M**= (**S**, **T**, Σ) be a GLAV schema mapping (i.e., Σ is a set of s-t tgds). Then, for every source instance I,

- The chase procedure produces a universal solution $\text{chase}_{\mathbf{M}}(I)$.
- The running time of the chase procedure is bounded by a polynomial in the size of I (PTIME data complexity).

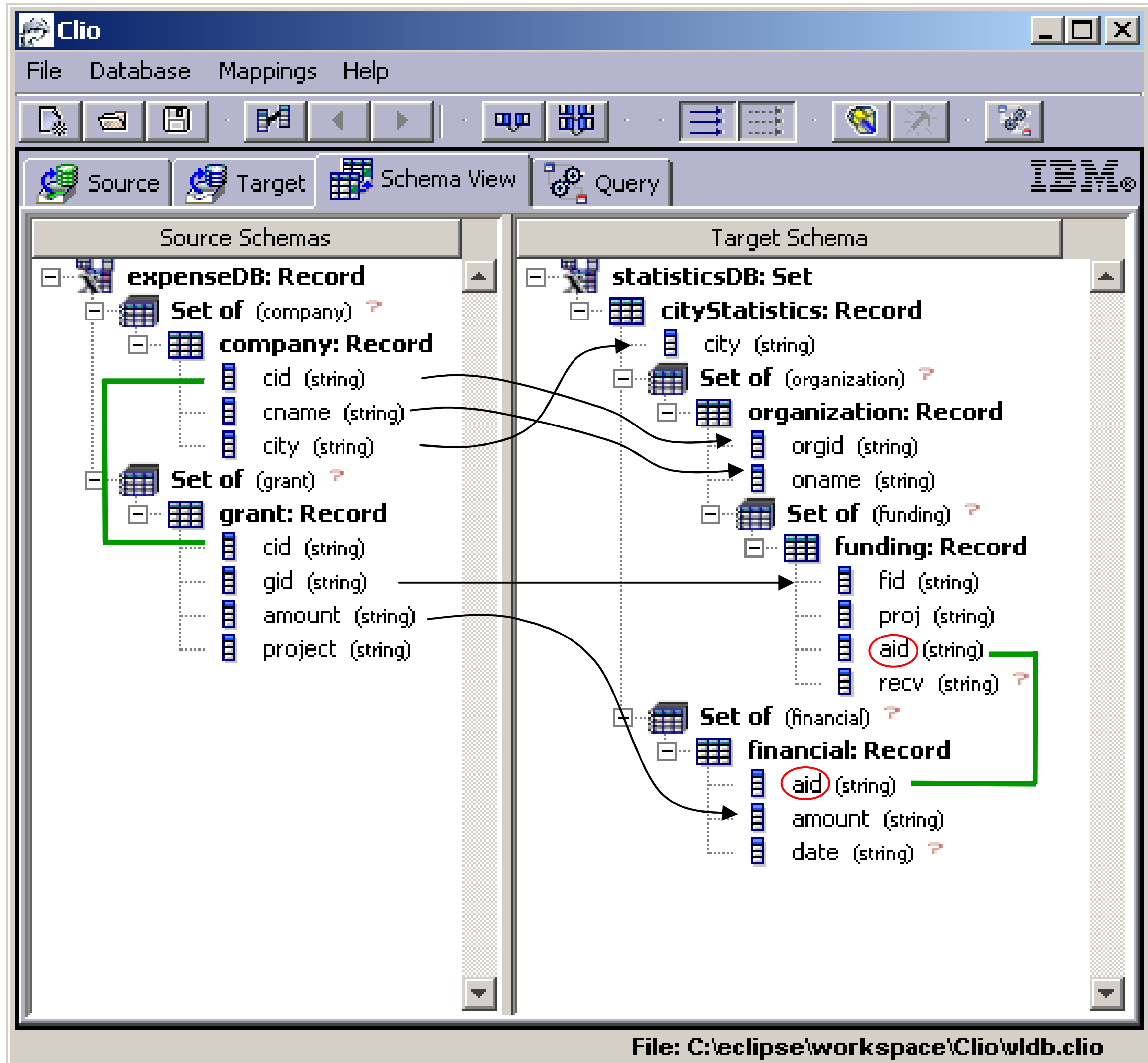
Note: The chase procedure can be used to produce universal solutions even in the presence of target constraints that obey certain mild structural conditions.

From Theory to Practice

- Clio Project at the IBM Almaden Research Center.
- Semi-automatic schema-mapping generation tool;
  - Data exchange system based on schema mappings.
- Universal solutions used as the semantics of data exchange.
- Universal solutions are generated via SQL queries extended with Skolem functions (implementation of chase procedure).
- Clio technology is now part of IBM Rational® Data Architect.

Some Features of Clio

- Supports nested structures
  - Nested Relational Model
  - Nested Constraints
- Automatic & semi-automatic discovery of attribute correspondence.
- Interactive derivation of schema mappings.
- Performs data exchange

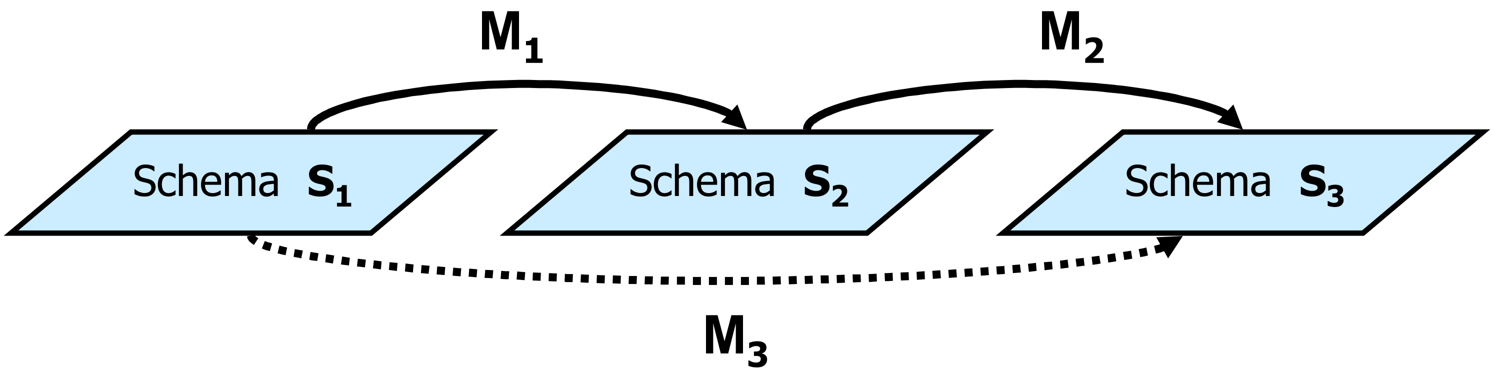


Managing Schema Mappings

- 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 schema-mapping operators, the most prominent of which are:
  - Composition operator
  - Inverse operator

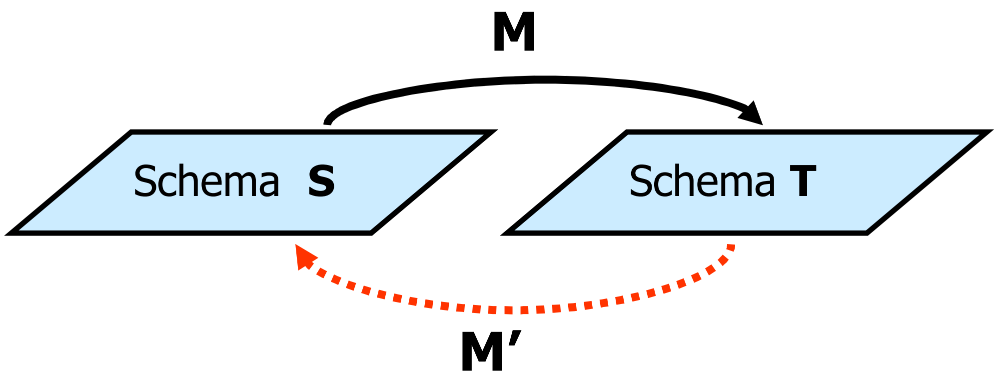
Composing Schema Mappings



- Given $\mathbf{M_1} = (\mathbf{S_1}, \mathbf{S_2}, \Sigma_1)$ and $\mathbf{M_2} = (\mathbf{S_2}, \mathbf{S_3}, \Sigma_2)$, derive a schema mapping $\mathbf{M_3} = (\mathbf{S_1}, \mathbf{S_3}, \Sigma_3)$ that is "equivalent" to the sequential application of $\mathbf{M_1}$ and $\mathbf{M_2}$.
- $\mathbf{M_3}$ is a **composition** of $\mathbf{M_1}$ and $\mathbf{M_2}$

$$\mathbf{M_3} = \mathbf{M_1} \circ \mathbf{M_2}$$

Inverting Schema Mapping

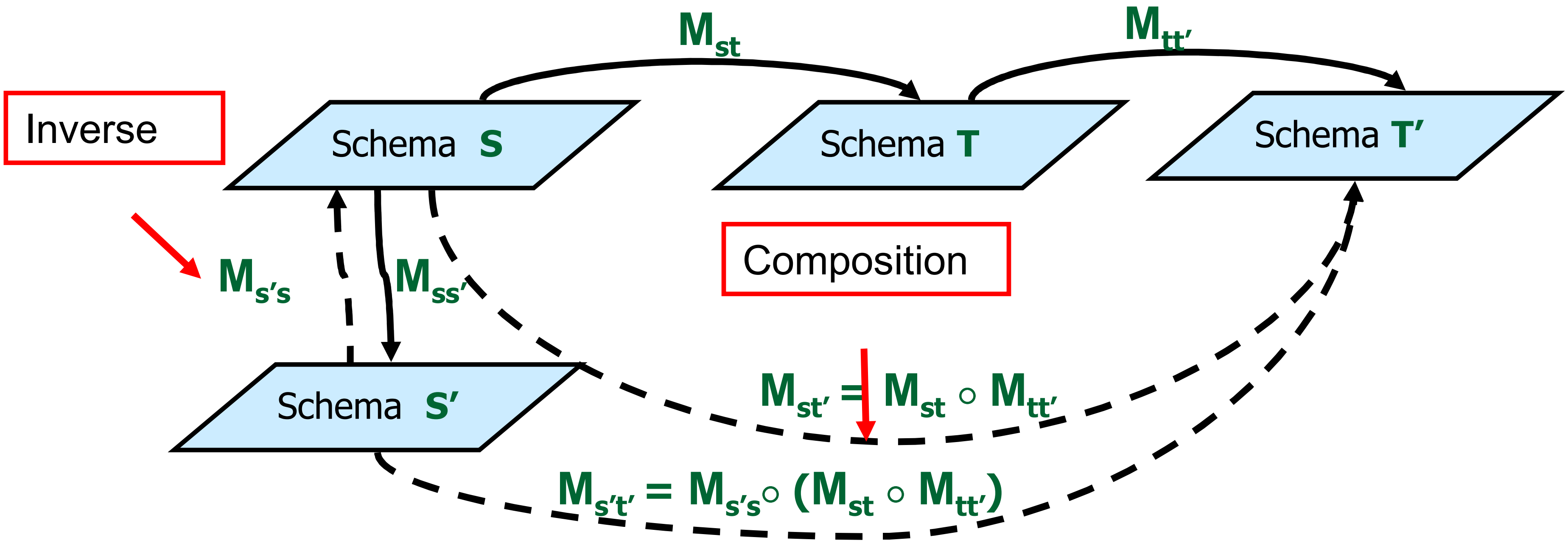


- Given **M**, derive **M'** that "undoes" **M**

M' is an **inverse** of **M**

- Composition and inverse can be applied to schema evolution.

Applications to Schema Evolution



Fact:

Schema evolution can be analyzed using the composition operator and the inverse operator.

Composing Schema Mappings

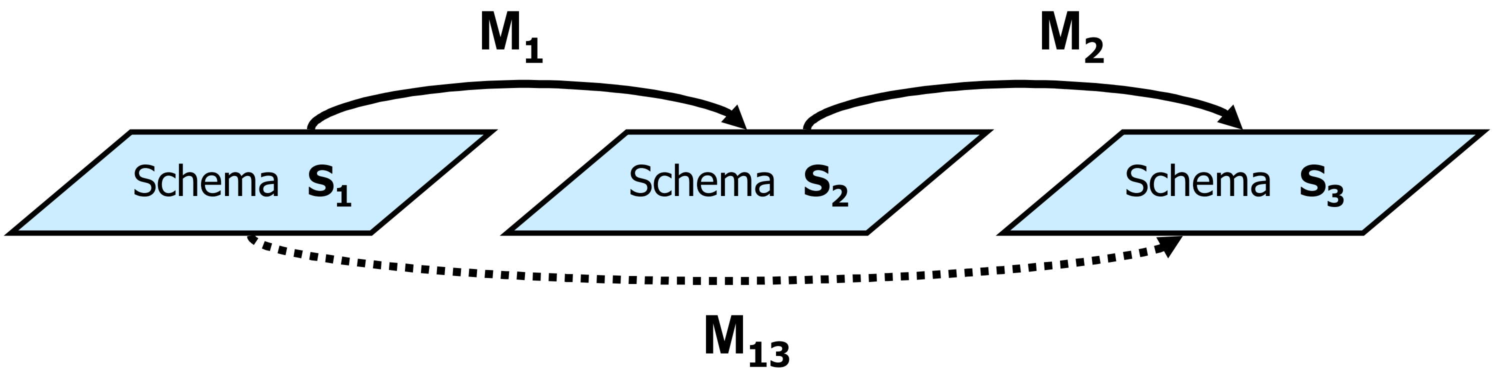
Main Issues:

- **Semantics**:
  What is the semantics of composition?

- **Language:**
  What is the language needed to express the composition of two schema mappings specified by s-t tgds?
  (GLAV schema mappings)

Note: Joint work with Fagin, Popa, and Tan

Composing Schema Mappings



- Given $\mathbf{M_1} = (\mathbf{S_1}, \mathbf{S_2}, \Sigma_1)$ and $\mathbf{M_2} = (\mathbf{S_2}, \mathbf{S_3}, \Sigma_2)$, derive a schema mapping $\mathbf{M_3} = (\mathbf{S_1}, \mathbf{S_3}, \Sigma_3)$ that is "equivalent" to the sequential application of $\mathbf{M_1}$ and $\mathbf{M_2}$.
- $\mathbf{M_3}$ is a **composition** of $\mathbf{M_1}$ and $\mathbf{M_2}$

$$\mathbf{M_3} = \mathbf{M_1} \circ \mathbf{M_2}$$

Semantics of Composition

- Recall that, from a semantic point of view, **M** can be identified with the binary relation

$$\text{Inst}(\mathbf{M}) = \{ (I,J): (I,J) \models \Sigma \}$$

- **Definition:**

A schema mapping $\mathbf{M_3}$ is a **composition** of $\mathbf{M_1}$ and $\mathbf{M_2}$ if

$$\text{Inst}(\mathbf{M_3}) = \text{Inst}(\mathbf{M_1}) \circ \text{Inst}(\mathbf{M_2}), \text{ that is,}$$

$$(I_1,I_3) \models \Sigma_3$$

if and only if

there exists I_2 such that $(I_1,I_2) \models \Sigma_1$ and $(I_2,I_3) \models \Sigma_2$.

The Composition of Schema Mappings

Fact: If both **M** = ($\mathbf{S}_1$, $\mathbf{S}_3$, Σ) and **M′** = ($\mathbf{S}_1$, $\mathbf{S}_3$, Σ') are compositions of $\mathbf{M_1}$ and $\mathbf{M_2}$, then Σ are Σ' are logically equivalent. For this reason:

- We say that **M** (or **M′**) is **the composition** of $\mathbf{M_1}$ and $\mathbf{M_2}$.
- We write $\mathbf{M_1} \circ \mathbf{M_2}$ to denote it

The Language of Composition: Good News

Theorem: Let $\mathbf{M_1}$ and $\mathbf{M_2}$ be consecutive schema mappings.

- If both $\mathbf{M_1}$ and $\mathbf{M_2}$ are GAV schema mappings, then their composition $\mathbf{M_1} \circ \mathbf{M_2}$ can be expressed as a GAV schema mapping.
- If $\mathbf{M_1}$ is a GAV schema mapping and $\mathbf{M_2}$ is a GLAV schema mappings, then their composition $\mathbf{M_1} \circ \mathbf{M_2}$ can be expressed as a GLAV schema mapping.

In symbols,

- GAV $\circ$ GAV = GAV
- GAV $\circ$ GLAV = GLAV

GAV $\circ$ GLAV = GLAV

Example:

- $\mathbf{M_1}$: GAV schema mapping

  Takes(s,m,c) $\rightarrow$ Student(s,m)

  Takes(s,m,c) $\rightarrow$ Enrolls(s,c)

- $\mathbf{M_2}$: GLAV schema mapping

  Student(s,m) $\wedge$ Enrolls(s,c) $\rightarrow$ $\exists$g Grade(s,m,c,g)

- $\mathbf{M_1} \circ \mathbf{M_2}$: GLAV schema mapping

  Takes(s,m,c) $\wedge$ Takes(s,m',c') $\rightarrow$ $\exists$g Grade(s,m,c',g)

The Language of Composition: Bad News

Theorem:

- GLAV schema mappings are **not** closed under composition.

  In symbols, GLAV $\circ$ GLAV $\not\subseteq$ GLAV.

- In fact, there is a LAV schema mapping $\mathbf{M_1}$ and a GAV schema mapping $\mathbf{M_2}$ such that $\mathbf{M_1} \circ \mathbf{M_2}$ is **not** expressible in least fixed-point logic LFP (hence, not in FO or in Datalog).

  In symbols, LAV $\circ$ GAV $\not\subseteq$ LFP.

LAV $\circ$ GAV $\not\subset$ LFP

- $\mathbf{M_1}$: LAV schema mapping

  $\forall x\, \forall y\, (E(x,y) \rightarrow \exists u \exists v\, (C(x,u) \wedge C(y,v)))$

  $\forall x\, \forall y\, (E(x,y) \rightarrow F(x,y))$
- $\mathbf{M_2}$: GAV schema mapping

  $\forall x\, \forall y\, \forall u\, \forall v\, (C(x,u) \wedge C(y,v) \wedge F(x,y) \rightarrow D(u,v))$

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

  Fact:

  G is 3-colorable if and only if $(I_1, I_3) \in$ Inst($\mathbf{M_1}$) $\circ$ Inst($\mathbf{M_2}$)

- **Theorem (Dawar – 1998):**

  3-Colorability is **not** expressible in LFP.

The Language of Composition

Question:

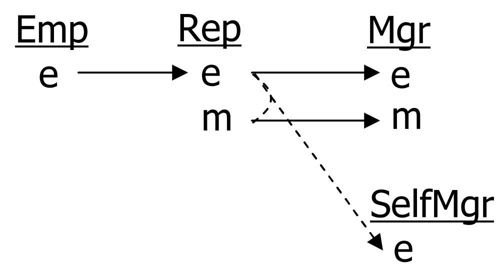
What is the “right” language for expressing the composition of two GLAV schema mappings?

Answer:

A fragment of **existential second-order logic** turns out to be the “right” language for this task.

Second-Order Logic to the Rescue

- **M_1** : LAV schema mapping
  - $\forall e\ (Emp(e) \rightarrow \exists m\ Rep(e,m))$
- **M_2** : GAV schema mapping
  - $\forall e\ \forall m\ (Rep(e,m) \rightarrow Mgr(e,m))$
  - $\forall e\ (Rep(e,e) \rightarrow SelfMgr(e))$



- **Theorem:** $\mathbf{M_1} \circ \mathbf{M_2}$ is **not** definable by **any** set (finite or infinite) of s-t tgds.

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

Second-Order Logic to the Rescue

- $\mathbf{M_1}$: LAV schema mapping
  - $\forall e\,(Emp(e) \rightarrow \exists m\, Rep(e,m))$
- $\mathbf{M_2}$: GAV schema mapping
  - $\forall e\, \forall m\,(Rep(e,m) \rightarrow Mgr(e,m))$
  - $\forall e\,(Rep(e,e) \rightarrow SelfMgr(e))$

- **Fact:** $\mathbf{M_1} \circ \mathbf{M_2}$ is expressible by the SO-tgd
  - $\exists \mathbf{f}\,(\forall e\,(Emp(e) \rightarrow Mgr(e,\mathbf{f(e)})) \wedge$
    $\forall e\,(Emp(e) \wedge (\mathbf{e=f(e)}) \rightarrow SelfMgr(e)))$.

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:

$\exists f_1 \ldots \exists f_m((\forall \mathbf{x_1}(\phi_1 \rightarrow \psi_1)) \wedge \ldots \wedge (\forall \mathbf{x}_n(\phi_n \rightarrow \psi_n)))$, where

- Each f_i is a function symbol.
- Each ϕ_i is a conjunction of atoms from **S** and equalities of terms.
- Each ψ_i is a conjunction of atoms from **T.**

Example: $\exists \mathbf{f}\ (\forall e(\ Emp(e) \rightarrow Mgr(e,\mathbf{f(e)})\ ) \wedge$
$\forall e(\ Emp(e) \wedge (\mathbf{e=f(e)}) \rightarrow SelfMgr(e)\ )\ )$

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 GLAV schema mappings. Hence, SO tgds are the "right" language for the composition of GLAV schema mappings.

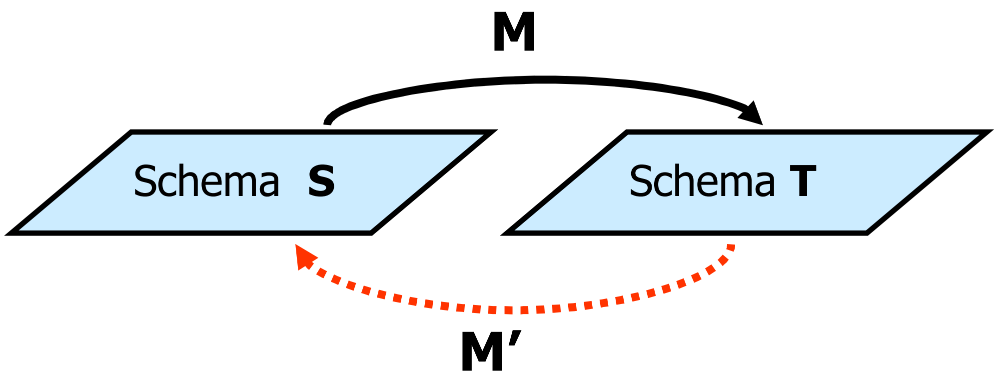
Synopsis of Schema Mapping Composition

- GAV $\circ$ GAV = GAV
- GAV $\circ$ GLAV = GLAV.
- GLAV $\circ$ GLAV $\not\subseteq$ GLAV. In fact, LAV $\circ$ GAV $\not\subseteq$ LFP.
- GLAV $\circ$ GLAV = SO-tgds = SO-tgds $\circ$ SO-tgds
  - SO-tgds are the "right" language for composing GLAV schema mappings.
  - SO-tgds are "chasable": Universal solutions in PTIME.
  - SO-tgds and the composition algorithm are supported in Clio.

Related Work (partial list)

- Earlier work on composition
  Madhavan and Halevy - 2003
- 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
- Composing schema mappings with target constraints
  Arenas, Fagin, Nash – 2010
- Composing LAV schema mappings with distinct variables
  Arocena, Fuxman, Miller - 2010

Inverting Schema Mapping



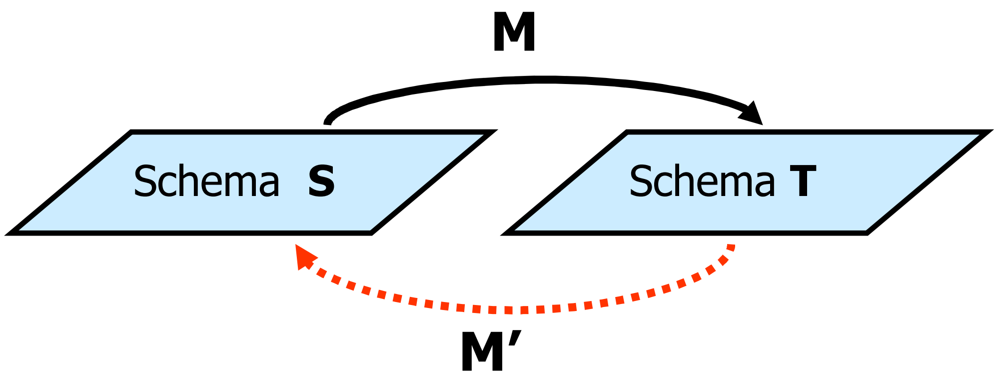
- Given **M**, derive **M'** that "undoes" **M**.
- **Question:**
  What is the "right" semantics of the inverse operator?
- **Note:**
  In general, **M** may have no "good" inverse, because **M** may have **information loss** (e.g., projection schema mapping).

The Semantics of the Inverse Operator

- Several different approaches:
  - (Exact) Inverses of schema mappings
    Fagin - 2006
  - Quasi-inverses of schema mappings
    Fagin, K …, Popa, Tan - 2007
  - Maximum recoveries of schema mappings
    Arenas, Pérez, Riveros - 2008
  - Extended maximum recoveries of schema mappings
    Fagin, K …, Popa, Tan – 2009

- **No** definitive semantics of the inverse operator has emerged.

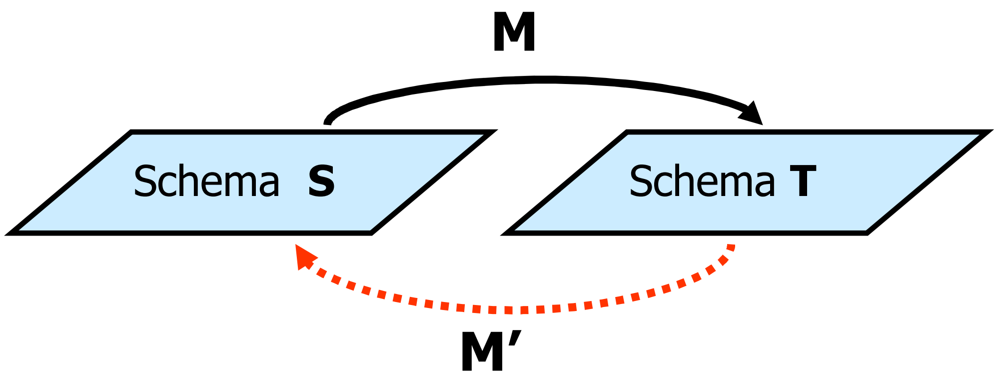
An Operational Approach to the Inverse



- Let I be an instance over **S**. Suppose that, after doing data exchange with **M**, this instance I is no longer available.
- An "inverse" schema mapping **M'** should be able to recover I to the extent possible.
- **Question:**

  How can this intuition be made precise?

Chasing and Chasing Back



Suppose that both **M** and **M'** are GLAV schema mappings.

- If I is an **S**-instance, then $chase_{\mathbf{M}}(I)$ is a **T**-instance.
- Consequently, $chase_{\mathbf{M'}}(chase_{\mathbf{M}}(I))$ is an **S**-instance.

Idea: Use $chase_{\mathbf{M'}}(chase_{\mathbf{M}}(I))$ to "recover" I.

Exact Chase-Inverse

Definition: Let **M** = (**S**, **T**, Σ) and **M'** = (**T**, **S**, Σ') be two GLAV schema mappings. **M'** is an **exact chase-inverse** of **M** if for every **S**-instance I, we have that I = $\text{chase}_{\mathbf{M'}}(\text{chase}_{\mathbf{M}}(I))$.

Example:

- **M** : Takes(s,m,c) $\rightarrow$ $\exists$n (Registers(s,m,n) $\wedge$ Course(n,c))
  (here n stands for the course number)
- **M'** : Registers(s,m,n) $\wedge$ Course(n,c) $\rightarrow$ Takes(s,m,c)

- **Fact: M'** is an exact chase-inverse of **M**.

Relaxing the Exact Chase-Inverse

- **M**: $E(x,y) \rightarrow \exists z\, (F(x,z) \wedge F(z,y))$

- **Fact:** **M** has **no** exact chase-inverse

  However, consider
- **M'**: $F(x,z) \wedge F(z,y) \rightarrow E(x,y)$
  If I = { E(1,2), E(2,3) }, then
  - $chase_{\mathbf{M}}(I)$ = { F(1,X), F(X,2), F(2,Y), F(Y,3) }
  - $chase_{\mathbf{M'}}(chase_{\mathbf{M}}(I))$ = { E(1,2), E(2,3), E(X,Y) }

- **Fact:** $I \approx_h chase_{\mathbf{M'}}(chase_{\mathbf{M}}(I))$, where $\approx_h$ denotes **homomorphic equivalence**. In fact, this holds for every I.

Chase-Inverse

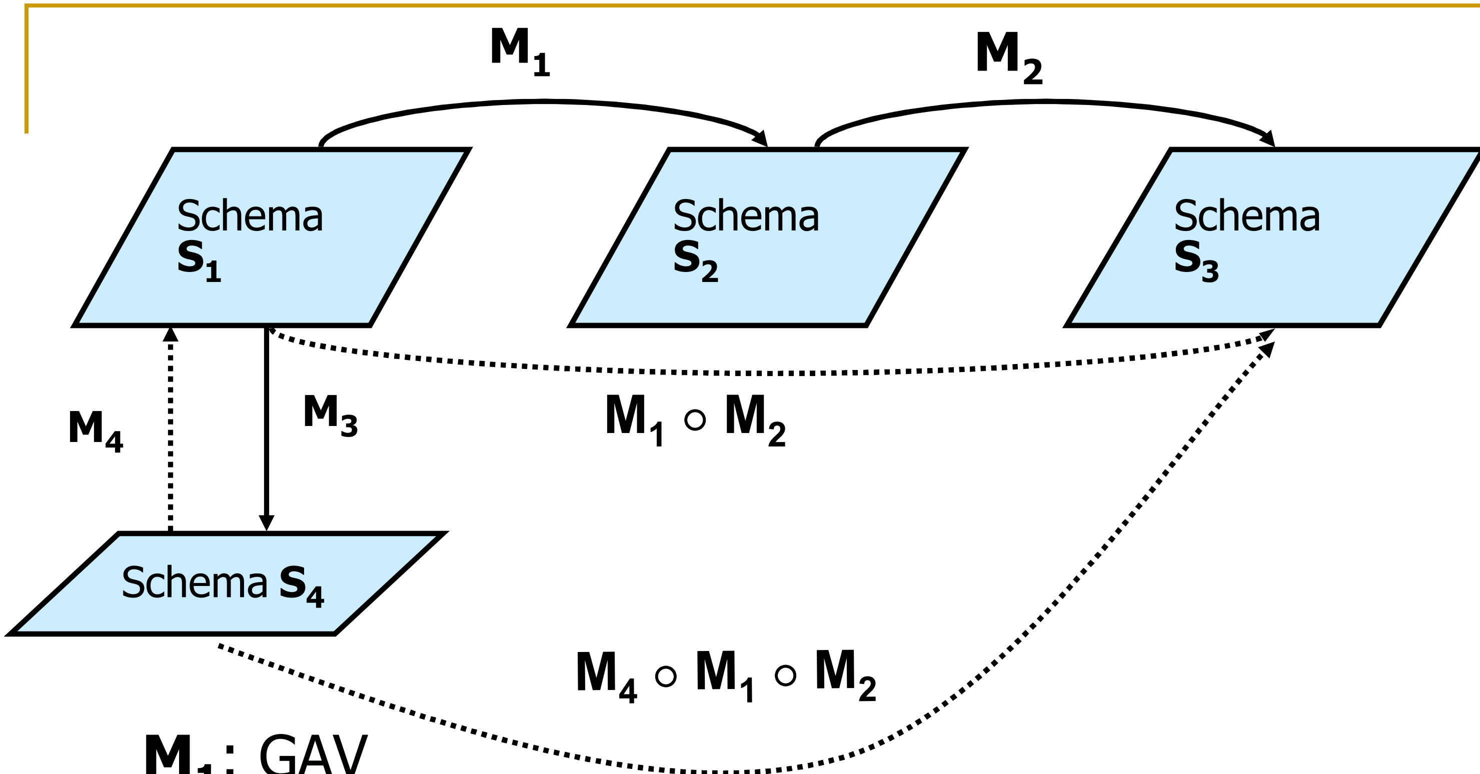
Definition: Let **M** = (**S**, **T**, Σ) and **M'** = (**T**, **S**, Σ') be two GLAV schema mappings. **M'** is a **chase-inverse** of **M** if

for every **S**-instance I, we have that $I \approx_h chase_{M'}(chase_M(I))$.

Note: A chase-inverse is good enough for data exchange purposes.

Theorem: If a GLAV schema mapping has a chase-inverse, then it has a GAV chase-inverse.

Note: This has nice implications for schema evolution.



$\mathbf{M_1}$: GAV

$\mathbf{M_2}$: GLAV

$\mathbf{M_3}$: GLAV

$\mathbf{M_4}$: GAV chase-inverse of $\mathbf{M_3}$.

Then the composition $\mathbf{M_4} \circ \mathbf{M_1} \circ \mathbf{M_2}$ is GLAV.

Much more remains to be done

- Pursue further the operational approach to inverses
  - Relaxations of chase-inverse
  - Language issues: disjunctive GLAV mappings arise naturally
  - Inverses of SO-tgds
  - …

- Applications of schema-mapping operators to:
  - Analysis of schema evolution.
  - Modeling and analysis of ETL (Extract-Transform-Load).