Foundations and Applications of Schema Mappings

Phokion G. Kolaitis

IBM Almaden Research Center
&
UC Santa Cruz

The Data Interoperability Problem

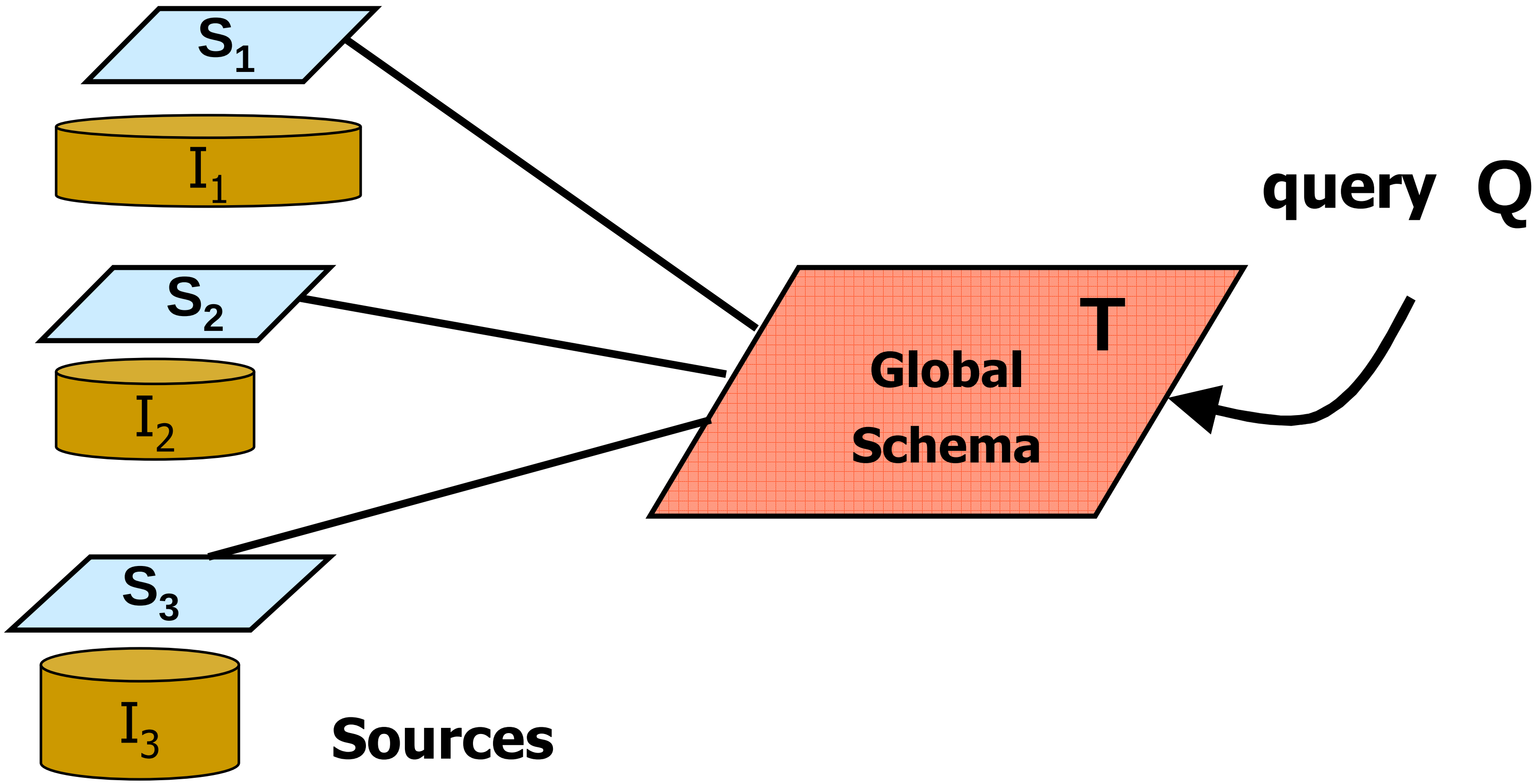
- Data may reside
  - at several different sites
  - in several different formats (relational, XML, …).

- Applications need to access all these data.

- Two different, but closely related, facets of data interoperability:
  - **Data Integration** (aka **Data Federation**):
  - **Data Exchange** (aka **Data Translation**):

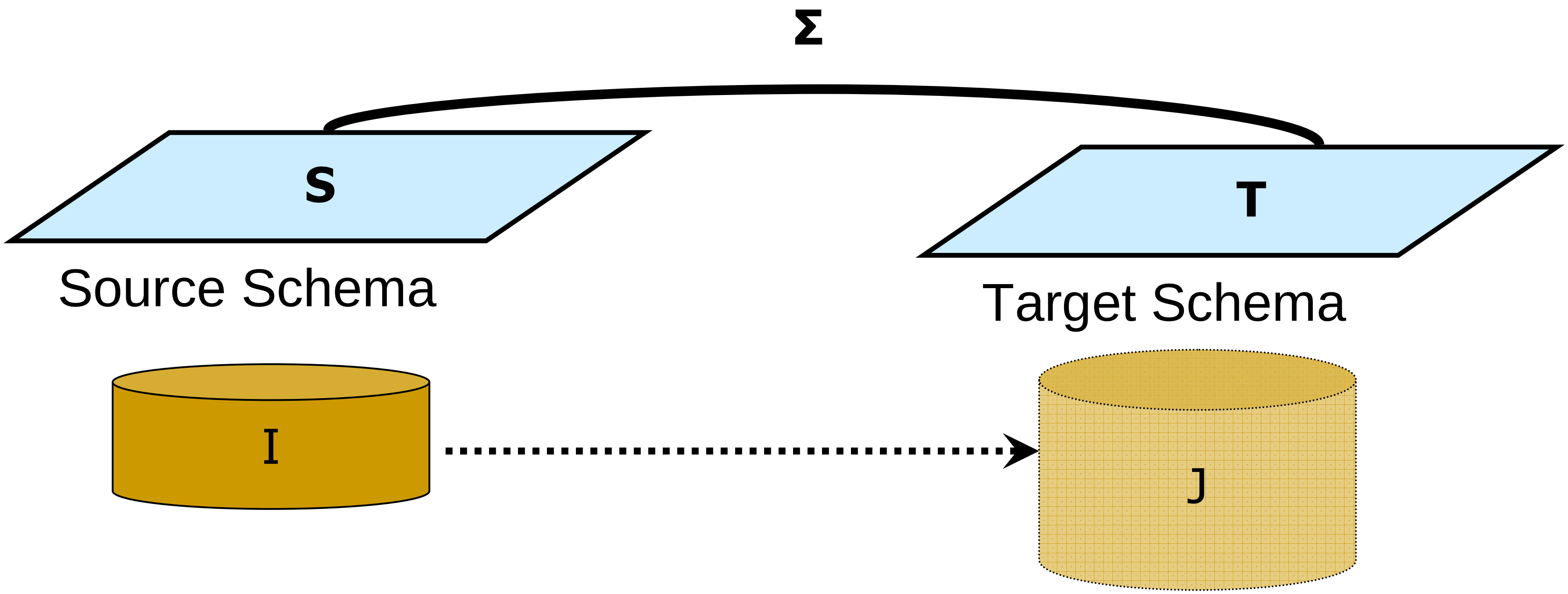
Data Integration

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.



Data Exchange

Data Exchange is an old, but recurrent, database problem

- Phil Bernstein – 2003
  "*Data exchange is the oldest database problem*"

- **EXPRESS**: IBM San Jose Research Lab – 1977
  EXtraction, **P**rocessing, and **RES**tructuring **S**ystem
  for transforming data between hierarchical databases.

- Data Exchange underlies:
  - Data Warehousing, ETL (Extract-Transform-Load) tasks;
  - XML Publishing, XML Storage, …

Challenges in Data Interoperability

Fact:

- Data interoperability tasks require expertise, effort, and time.
- Human experts have to generate complex transformations that specify the relationship between schemas written as programs (e.g., in Java) or as SQL/XSLT scripts.
- At present, there is relatively little automation.

Question: How can we address these challenges?

Answer: Introduce a higher level of abstraction that makes it possible to separate the design of the relationship between schemas from its implementation.

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 help with the development of tools:
  - Are easier to generate and manage (semi)-automatically;
  - Can be compiled into SQL/XSLT scripts automatically.

Outline of the Tutorial

- Schema Mappings as a framework for formalizing and studying data interoperability tasks.

- Data Exchange and Solutions in Data Exchange
  - Universal Solutions and the Core.

- Query Answering in Data Exchange.

- Managing schema mappings via operators:
  - The composition operator
  - The inverse operator and its variants

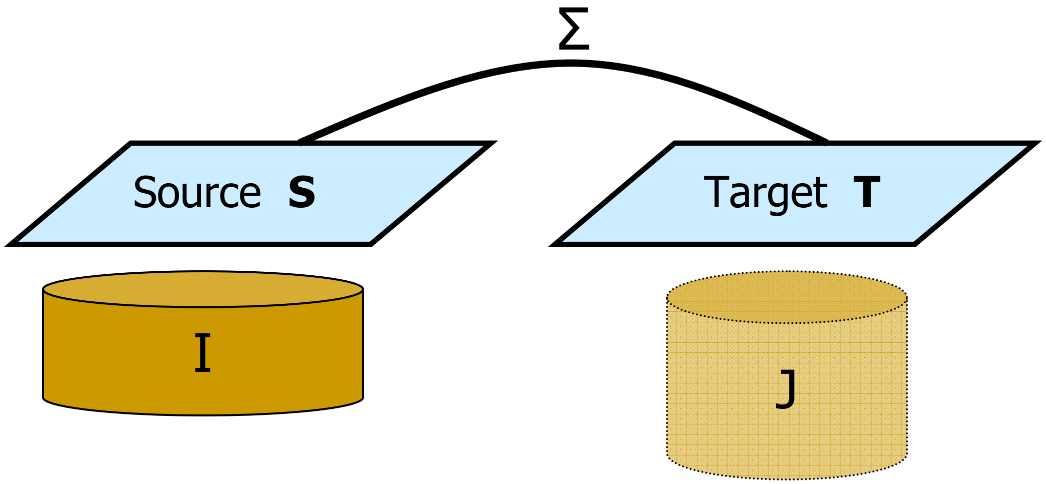
Acknowledgments

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

  Papers in ICDT 2003, PODS 2003-2008, 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.

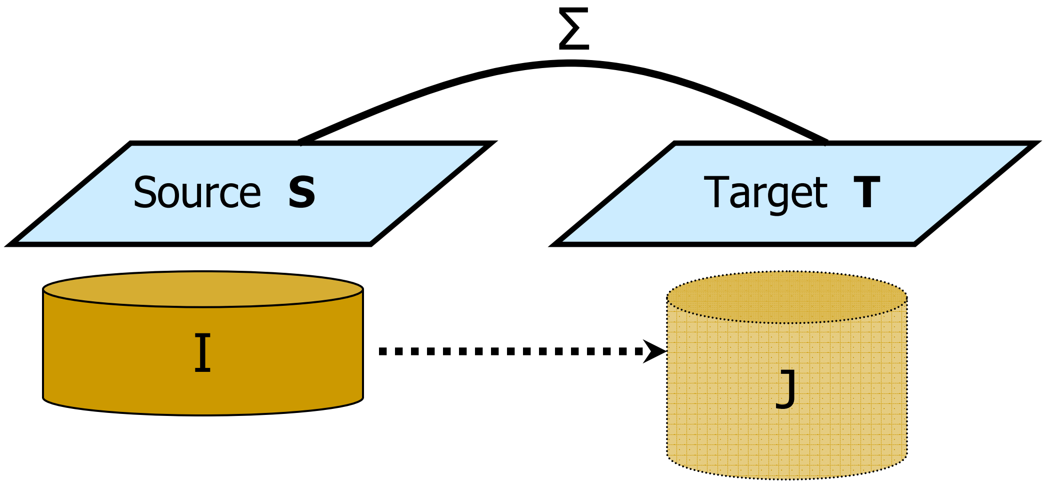
Schema Mappings



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

- Inst(**M**) = { (I, J): I is an **S**-instance, J is a **T**-instance, and (I, J) ⊨ Σ }.

Schema Mappings & Data Exchange



- 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**.
- Data Exchange via the schema mapping **M** = (**S**, **T**, Σ)

  Transform a given source instance I to a target instance J, so that (I, J) satisfy the specifications Σ of **M**.

Solutions in Schema Mappings

Definition: Schema Mapping **M** = (**S**, **T**, Σ)

If I is a source instance, then a solution for I is a target instance J such that (I, J) satisfy Σ.

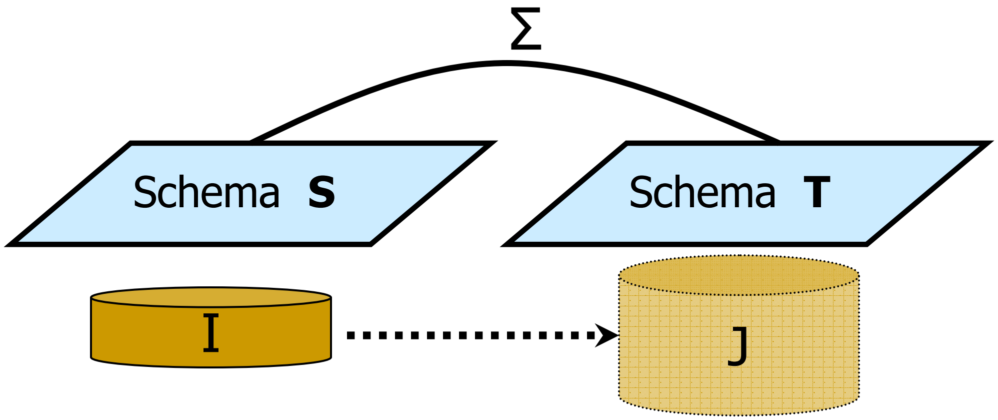
Fact: In general, for a given source instance I,

- No solution for I may exist

  or

- Multiple solutions for I may exist; in fact, infinitely many solutions for I may exist.

Schema Mappings: Basic Problems



Definition: Schema Mapping **M** = (**S**, **T**, Σ)

- The existence-of-solutions problem **Sol(M)**: (decision problem)
  Given a source instance I, is there a solution J for I?

- The data exchange problem associated with **M**: (function problem)
  Given a source instance I, construct a solution J for I, provided a solution exists.

Schema Mapping Specification Languages

- Ideally, schema mappings should be
  - expressive enough to specify data interoperability tasks;
  - simple enough to be efficiently manipulated by tools.

- **Question**: How are schema mappings specified?

- **Answer**: Use logic. In particular, it is natural to try to use first-order logic as a specification language for schema mappings.

- **Fact**: There is a fixed first-order sentence specifying a schema mapping **M*** such that **Sol(M*)** is undecidable.

- Hence, we need to restrict ourselves to well-behaved fragments of first-order logic.

Embedded Implicational Dependencies

- Dependency Theory: extensive study of constraints in relational databases in the 1970s and 1980s.

- Embedded Implicational Dependencies: Fagin, Beeri-Vardi, …

  Class of constraints with a balance between high expressive power and good algorithmic properties:

  - Tuple-generating dependencies (tgds)

  Inclusion and multi-valued dependencies are a special case.

  - Equality-generating dependencies (egds)

  Functional dependencies are a special case.

Schema Mapping Specification Language

The relationship between source and target is given by formulas of first-order logic, called

Source-to-Target Tuple Generating Dependencies (s-t tgds)

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

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

Example:

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

Schema Mapping Specification Language

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

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

- s-t tgds generalize the main specifications used in data integration:
  - They generalize LAV (local-as-view) specifications:

    $P(\mathbf{x}) \rightarrow \exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y})$, where P is a source schema.
  - They generalize GAV (global-as-view) specifications:

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

    (they are equivalent to full tgds: $\varphi(\mathbf{x}) \rightarrow \psi(\mathbf{x})$,

    where $\varphi(\mathbf{x})$ and $\psi(\mathbf{x})$ are conjunctions of atoms).

Target Dependencies

In addition to source-to-target dependencies, we also consider target dependencies:

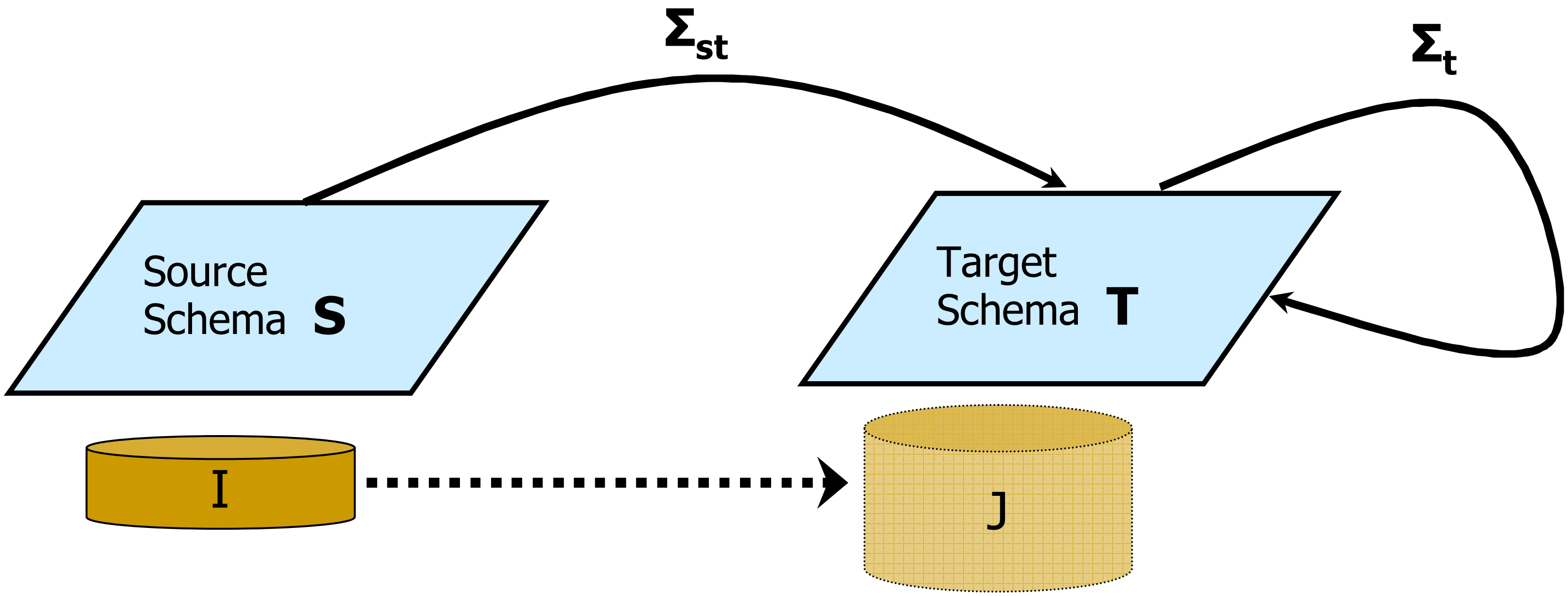
- Target Tgds : $\varphi_T(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi_T(\mathbf{x}, \mathbf{y})$

  Dept (did, dname, mgr_id, mgr_name) $\rightarrow$ Mgr (mgr_id, did)
  (a target inclusion dependency constraint)

- Target Equality Generating Dependencies (egds):
  $$\varphi_T(\mathbf{x}) \rightarrow (x_1 = x_2)$$

  $(\text{Mgr}(e, d_1) \wedge \text{Mgr}(e, d_2)) \rightarrow (d_1 = d_2)$
  (a target key constraint)

Data Exchange Framework



Schema Mapping **M** = (**S**, **T**, Σ_{st} , Σ_t), where

- Σ_{st} is a set of source-to-target tgds
- Σ_t is a set of target tgds and target egds

Underspecification in Data Exchange

- **Fact:** Given a source instance, multiple solutions may exist.

- **Example:**

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

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

  Source instance I = {E(a,b)}

  Solutions: Infinitely many solutions exist

  - J_1 = {H(a,b), H(b,b)}
  - J_2 = {H(a,a), H(a,b)}
  - J_3 = {H(a,X), H(X,b)}
  - J_4 = {H(a,X), H(X,b), H(a,Y), H(Y,b)}
  - J_5 = {H(a,X), H(X,b), H(Y,Y)}

constants:

a, b, …

variables (labelled nulls):

X, Y, …

Main issues in data exchange

For a given source instance, there may be multiple target instances satisfying the specifications of the schema mapping. Thus,

- When more than one solution exist, which solutions are "better" than others?
- How do we compute a "best" solution?
- In other words, what is the "right" semantics of data exchange?

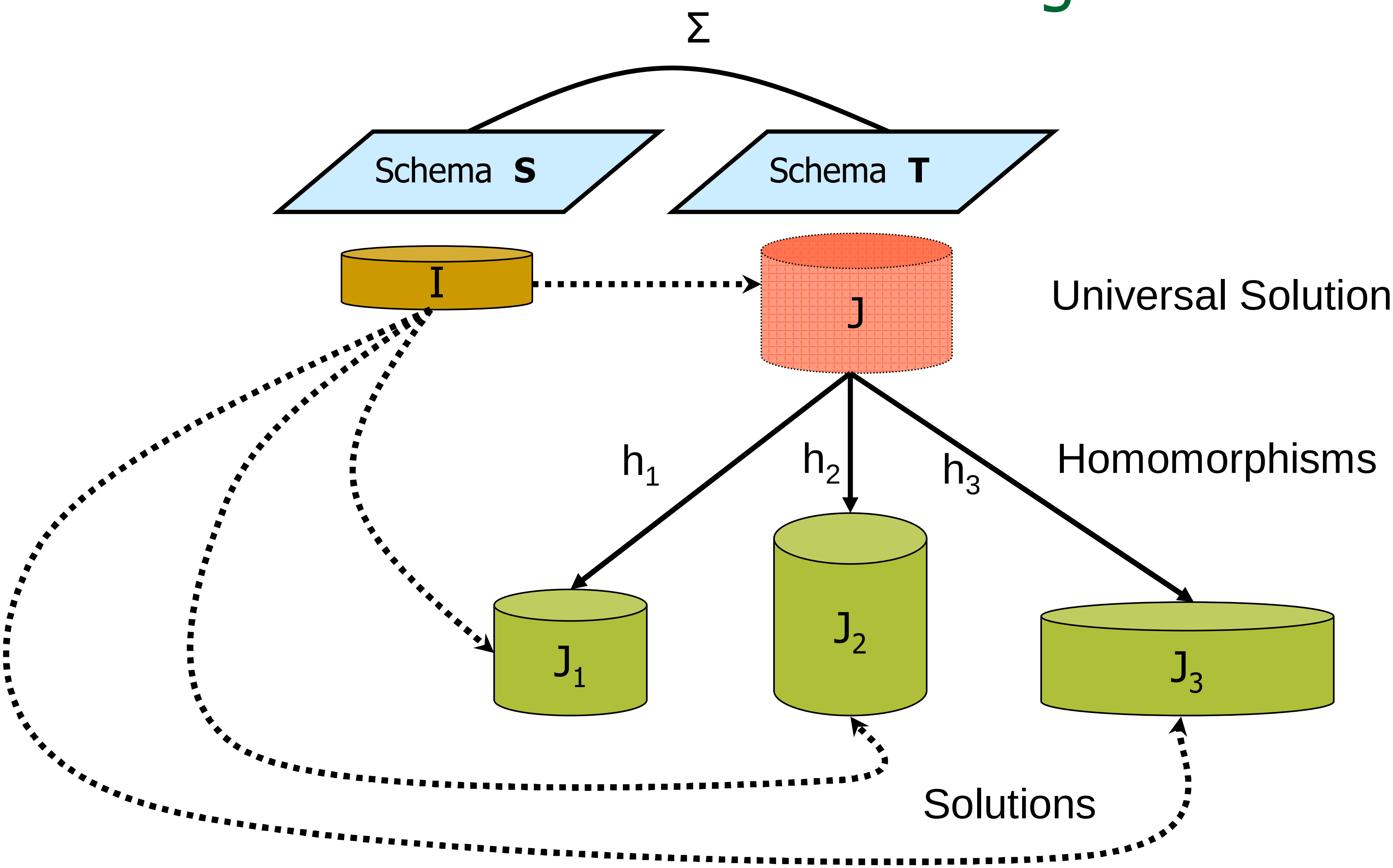
Universal Solutions in Data Exchange

Definition (FKMP 2003): A solution is universal if it has homomorphisms to all other solutions (thus, it is a "most general" solution).

- Constants: entries in source instances
- Variables (labeled nulls): other entries in target instances
- Homomorphism h: $J_1 \rightarrow J_2$ between target instances:
  - h(c) = c, for constant c
  - If $P(a_1,\ldots,a_m)$ is in J_1, then $P(h(a_1),\ldots,h(a_m))$ is in J_2.

Claim: Universal solutions are the *preferred* solutions in data exchange.

Universal Solutions in Data Exchange



Example - continued

Source relation S(A,B), target relation T(A,B)

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

Source instance I = {H(a,b)}

Solutions: Infinitely many solutions exist

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

Structural Properties of Universal Solutions

- Universal solutions are analogous to most general unifiers in logic programming.

- Uniqueness up to homomorphic equivalence:
  If J and J' are universal for I, then they are homomorphically equivalent.

- Representation of the entire space of solutions:
  Assume that J is universal for I, and J' is universal for I'.
  Then the following are equivalent:
  1. I and I' have the same space of solutions.
  2. J and J' are homomorphically equivalent.

The Existence-of-Solutions Problem

Question: What can we say about the existence-of-solutions problem **Sol(M)** for a fixed schema mapping **M** = (**S**, **T**, Σ_{st},Σ_t) specified by s-t tgds and target tgds and egds?

Answer: Depending on the target constraints in Σ_t:

- **Sol(M)** can be trivial (solutions always exist).

  …
- **Sol(M)** can be in PTIME.

  …
- **Sol(M)** can be undecidable.

Algorithmic Problems in Data Exchange

Proposition: If **M** = (**S**, **T**, Σ_{st}, Σ_t) is a schema mapping such that Σ_t is a set of **full target tgds**, then:

- Solutions always exist; hence, **Sol(M)** is trivial.
- There is a **Datalog program** π over the target **T** that can be used to compute universal solutions as follows:
  Given a source instance I,
  1. Compute a universal solution J* for I w.r.t. the schema mapping **M*** = (**S**, **T**, Σ_{st}) using the **naïve chase** algorithm.
  2. Run the **Datalog program** π on J* to obtain a universal solution J for I w.r.t. **M**.
- Consequently, universal solutions can be computed in polynomial time.

Algorithmic Problems in Data Exchange

Naïve Chase Algorithm for **M*** = (**S**, **T**, Σ_{st}) : given a source instance I, build a target instance J* that satisfies each s-t tgd in Σ_{st}

- by introducing new facts in J as dictated by the RHS of the s-t tgd

and

- by introducing new values (variables) in J each time existential quantifiers need witnesses.

Example: **M** = (**S**, **T**, Σ_{st}, Σ_t)

$$\Sigma_{st}: \; E(x,y) \rightarrow \exists z(F(x,z) \wedge F(z,y))$$

$$\Sigma_t: \; F(u,w) \wedge F(w,v) \rightarrow F(u,v)$$

1. The naïve chase returns a relation F* obtained from E by adding a new node between every edge of E.
2. The Datalog program π computes the **transitive closure** of F*.

Algorithmic Problems in Data Exchang

Proposition : If **M** = (**S**, **T**, Σ_{st},Σ_t) is a schema mapping such that Σ_t is a set of **full target tgds** and **target egds**, then:

- Solutions need not always exist.
- The existence-of-solutions problem **Sol(M)** is in PTIME, and may be PTIME-complete.

Proof: Reduction from Horn 3-SAT.

Algorithmic Problems in Data Exchange

Reducing Horn 3-SAT to the Existence-of-Solutions Problem **Sol(M)**

- Σ_{st}:
  $U(x) \rightarrow U'(x)$
  $P(x,y,z) \rightarrow P'(x,y,z)$
  $N(x,y,z) \rightarrow N'(x,y,z)$
  $V(x) \rightarrow V'(x)$

- Σ_{t}:
  $U'(x) \rightarrow M'(x)$
  $P'(x,y,z) \wedge M'(y) \wedge M'(z) \rightarrow M'(x)$
  $N'(x,y,z) \wedge M'(x) \wedge M'(y) \wedge M'(z) \wedge V'(u) \rightarrow W'(u)$
  $W'(u) \wedge W'(v) \rightarrow u = v$

- $U(x)$ encodes the unit clause x
  $P(x,y,z)$ encodes the clause $(\neg y \vee \neg z \vee x)$
  $N(x,y,z)$ encodes the clause $(\neg x \vee \neg y \vee \neg z)$
  $V = \{0, 1\}$

Algorithmic Problems in Data Exchange

Question:

What about arbitrary target tgds and egds?

Undecidability in Data Exchange

Theorem (K …, Panttaja, Tan - 2006):

There is a schema mapping **M**= (**S**, **T**, Σ^*_{st}, Σ^*_t) such that:

- Σ^*_{st} consists of a single source-to-target tgd;
- Σ^*_t consists of one egd, one full target tgd, and one (non-full) target tgd;
- The existence-of-solutions problem **Sol(M)** is undecidable.

Hint of Proof:

Reduction from the

Embedding Problem for Finite Semigroups:

Given a finite partial semigroup, can it be embedded to a finite semigroup?

The Embedding Problem & Data Exchange

Reducing the **Embedding Problem for Semigroups** to **Sol(M)**

- Σ_{st}: $R(x,y,z) \rightarrow R'(x,y,z)$

- Σ_t:
  - R′ is a partial function:
    $R'(x,y,z) \wedge R'(x,y,w) \rightarrow z = w$

  - R′ is associative
    $R'(x,y,u) \wedge R'(y,z,v) \wedge R'(u,z,w) \rightarrow R'(x,u,w)$

  - R′ is a total function
    $$R'(x,y,z) \wedge R'(x',y',z') \rightarrow \exists w_1 \ldots \exists w_9$$
    $$(R'(x,x',w_1) \wedge R'(x,y',w_2) \wedge R'(x,z',w_3)$$
    $$R'(y,x',w_4) \wedge R'(y,y',w_5) \wedge R'(x,z',w_6)$$
    $$R'(z,x',w_7) \wedge R'(z,y',w_8) \wedge R'(z,z',w_9))$$

The Existence-of-Solutions Problem

Summary: The existence-of-solutions problem

- is **undecidable** for schema mappings in which the target dependencies are arbitrary tgds and egds;
- is in **PTIME** for schema mappings in which the target dependencies are **full** tgds and egds.

Question: Are there classes of target tgds **richer** than full tgds and egds for which the existence-of-solutions problem is in **PTIME**?

Algorithmic Properties of Universal Solutions

Theorem (FKMP 2003): Schema mapping **M**= (**S**, **T**, Σ_{st}, Σ_t) such that:

- Σ_{st} is a set of source-to-target tgds;
- Σ_t is the union of a weakly acyclic set of target tgds with a set of target egds.

Then:

- Universal solutions exist if and only if solutions exist.
- **Sol(M)** is in PTIME.
- A *canonical* universal solution (if a solution exists) can be produced in polynomial time using the chase procedure.

Weakly Acyclic Sets of Tgds

Weakly acyclic sets of tgds contain as special cases:

- **Sets of full tgds**

  $$\varphi_T(\mathbf{x},\mathbf{x'}) \rightarrow \psi_T(\mathbf{x}),$$

  where $\varphi_T(\mathbf{x},\mathbf{x'})$ and $\psi_T(\mathbf{x})$ are conjunctions of target atoms.

- **Acyclic sets of inclusion dependencies**

  Large class of dependencies occurring in practice.

Weakly Acyclic Sets of Tgds: Definition

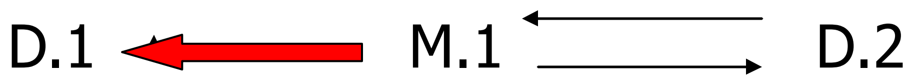
- **Position graph** of a set Σ of tgds:
  - **Nodes:** R.A, with R relation symbol, A attribute of R
  - **Edges:** for every $\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})$ in Σ, for every x in $\mathbf{x}$ occurring in ψ, for every occurrence of x in φ in R.A:
    - For every occurrence of x in ψ in S.B,
      add an edge R.A $\longrightarrow$ S.B
    - In addition, for every existentially quantified y that occurs in ψ in T.C, add a **special edge** R.A $\Longrightarrow$ T.C

- Σ is **weakly acyclic** if the position graph has **no** cycle containing a **special edge**.

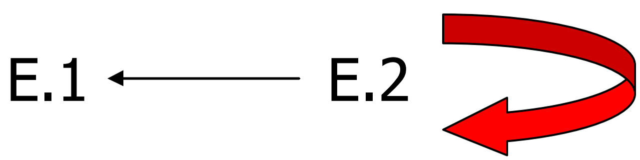
- A tgd θ is **weakly acyclic** if so is the singleton set $\{\theta\}$.

Weakly Acyclic Sets of Tgds: Examples

- **Example 1:** $\{ D(e,m) \rightarrow M(m),\ \ M(m) \rightarrow \exists\, e\ D(e,m) \}$
  is weakly acyclic, but cyclic.



- **Example 2:** $\{ E(x,y) \rightarrow \exists\, z\ E(y,z) \}$
  is not weakly acyclic.



Data Exchange with Weakly Acyclic Tgds

Theorem (FKMP): Schema mapping **M**= (**S**, **T**, Σ_{st}, Σ_t) such that:

- Σ_{st} is a set of source-to-target tgds;
- Σ_t is the union of a weakly acyclic set of target tgds with a set of target egds.

There is an algorithm, based on the chase procedure, so that:

- Given a source instance I, the algorithm determines if a solution for I exists; if so, it produces a canonical universal solution for I.
- The running time of the algorithm is polynomial in the size of I.
- Hence, the existence-of-solutions problem **Sol(M)** for **M**, is in PTIME.

Chase Procedure for Tgds and Egds

Given a source instance I,

1. Use the naïve chase to chase I with Σ_{st} and obtain a target instance J*.

2. Chase J * with the target tgds and the target egds in Σ_t to obtain a target instance J as follows:

2.1. For target tgds introduce new facts in J as dictated by the RHS of the s-t tgd and introduce new values (variables) in J each time existential quantifiers need witnesses.

2.2. For target egds $\phi(x) \rightarrow x_1 = x_2$

2.2.1. If a variable is equated to a constant, replace the variable by that constant;

2.2.2. If one variable is equated to another variable, replace one variable by the other variable.

2.2.3 If one constant is equated to a different constant, stop and repor "failure".

The Existence of Solutions Problem

Summary: The existence-of-solutions problem

- is undecidable for schema mappings in which the target dependencies are arbitrary tgds and egds;
- is in PTIME for schema mappings in which the set of the target dependencies is the union of a weakly acyclic set of tgds and a set of egds.

Note:

- These are data complexity results.
- The combined complexity of the existence-of-solutions problem is 2EXPTIME-complete (weakly acyclic sets of target tgds and egds).

The Smallest Universal Solution

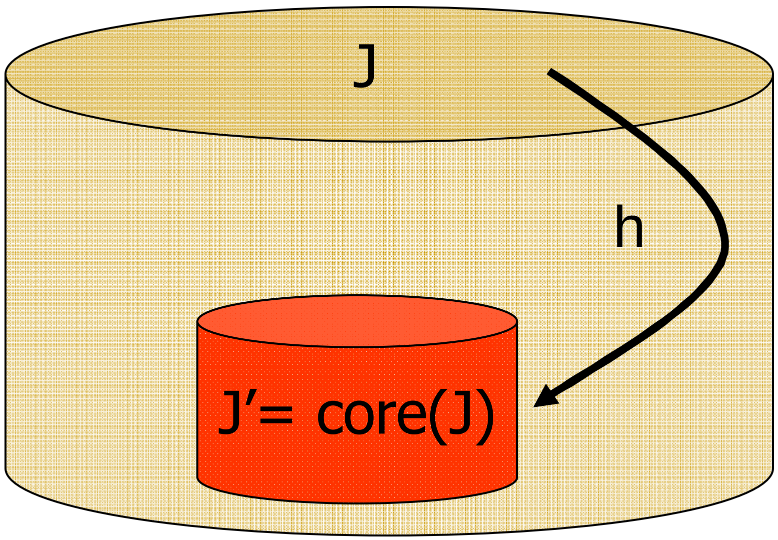
- **Fact:** Universal solutions need not be unique.
- **Question**: Is there a "best" universal solution?
- **Answer:** In joint work with R. Fagin and L. Popa, we took a "small is beautiful" approach:

  There is a smallest universal solution (if solutions exist); hence, the most compact one to materialize.

- **Definition:** The core of an instance J is the smallest subinstance J' that is homomorphically equivalent to J.

- **Fact:**
  - Every finite relational structure has a core.
  - The core is unique up to isomorphism.

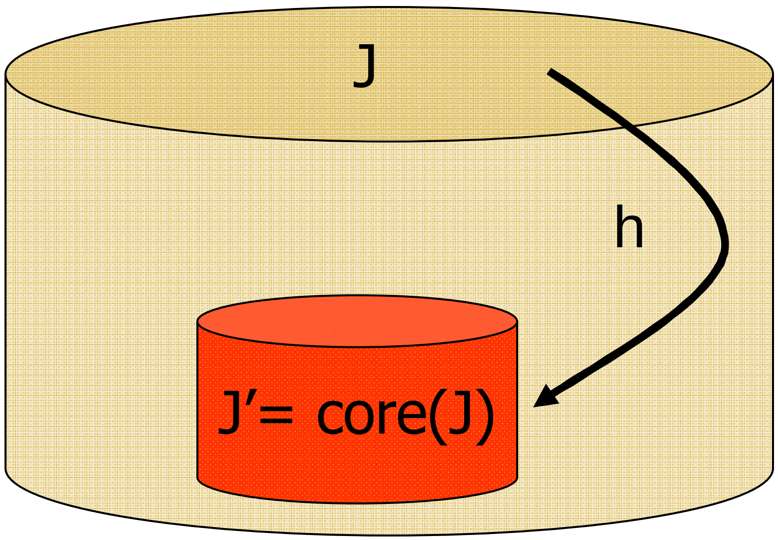
The Core of a Structure



Definition: J′ is the core of J if

- J′ ⊆ J
- there is a hom. h: J → J′
- there is **no** hom. g: J → J″, where J″ ⊂ J′.

The Core of a Structure

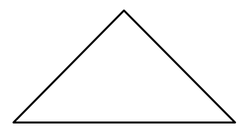


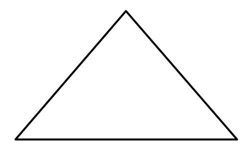
Definition: J′ is the core of J if

- $J' \subseteq J$

- there is a hom. $h: J \rightarrow J'$

- there is **no** hom. $g: J \rightarrow J''$, where $J'' \subset J'$.

Example: If a graph **G** contains a , then

G is 3-colorable if and only if core(**G**) =  .

Fact: Computing cores of graphs is an NP-hard problem.

Example - continued

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

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

Source instance I = {E(a,b)}.

Solutions: Infinitely many universal solutions exist.

- J_3 = {H(a,X), H(X,b)} is the core.
- J_4 = {H(a,X), H(X,b), H(a,Y), H(Y,b)} is universal, but not the core.
- J_5 = {H(a,X), H(X,b), H(Y,Y)} is not universal.

Core: The smallest universal solution

Theorem (FKP 2003): **M** = (**S**, **T**, Σ_{st}, Σ_t) a schema mapping:

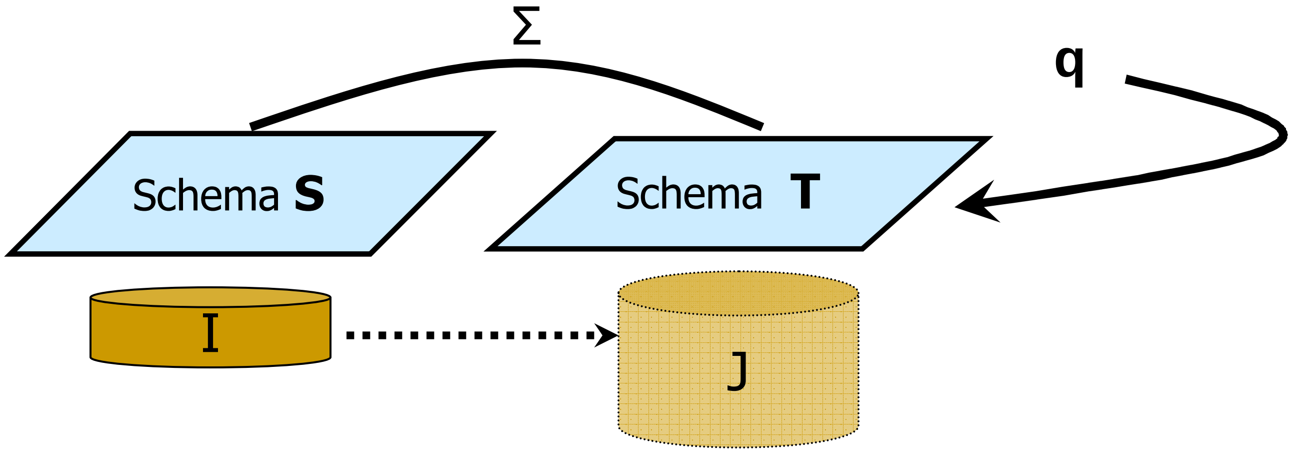
- All universal solutions have the same core.
- The core of the universal solutions is the smallest universal solution.
- If every target constraint is an egd, then the core is polynomial-time computable.

Theorem (Nash & Gottlob 2006): Let **M** = (**S**, **T**, Σ_{st}, Σ_t) be such that Σ_t is the union of a set of weakly acyclic target tgds with a set of target egds. Then the core is polynomial-time computable.

Outline of the Tutorial

- ✓ Schema Mappings and Data Exchange
- ✓ Solutions in Data Exchange
  - ✓ Universal Solutions
  - ✓ The Core of the Universal Solutions
- Query Answering in Data Exchange
- Composing and Inverting Schema Mappings.

Query Answering in Data Exchange



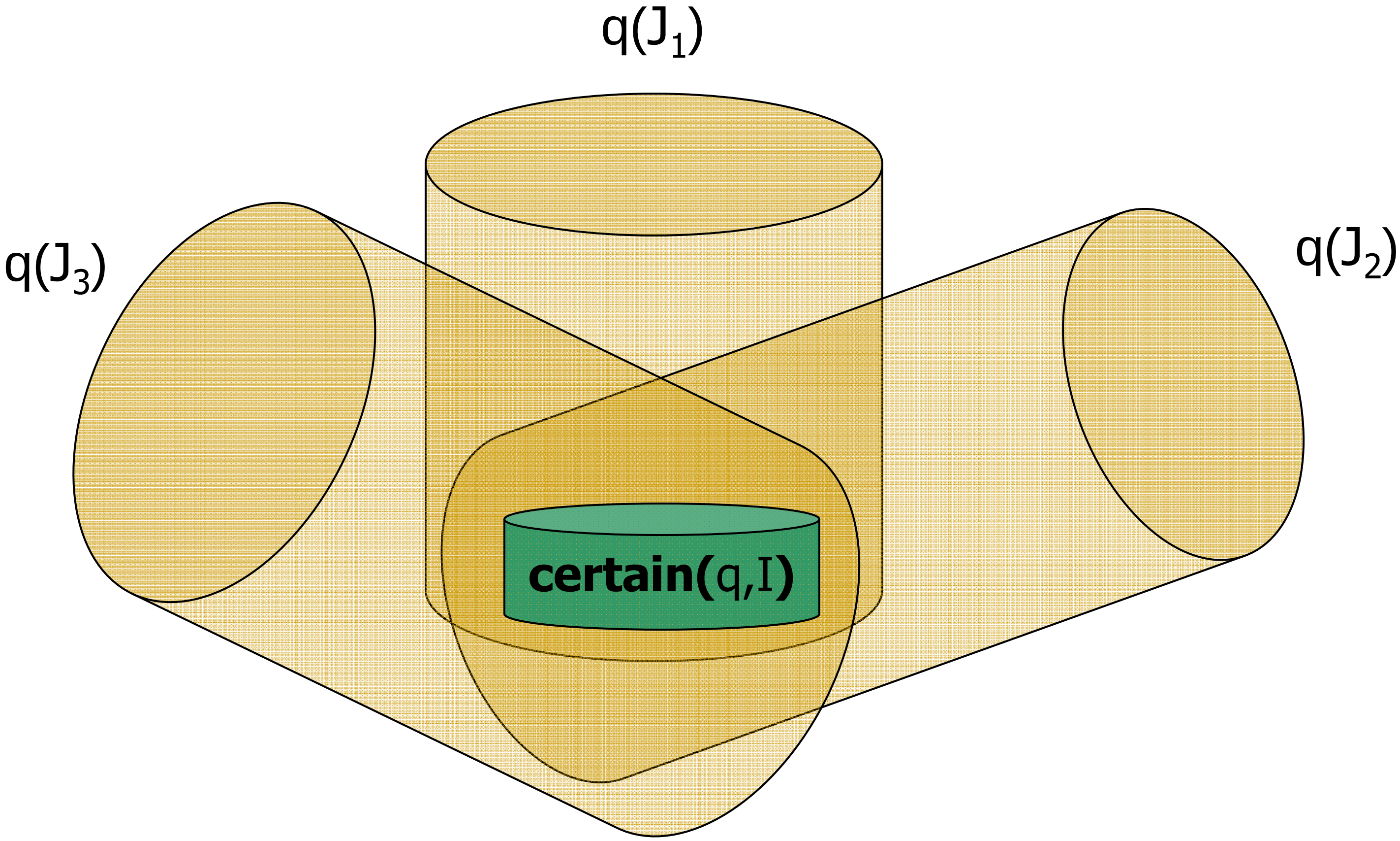
Question: What is the semantics of target query answering?

Definition: The certain answers of a query q over **T** on I

$$\textbf{certain}(q,I) = \bigcap \{\, q(J):\ J \text{ is a solution for } I \,\}.$$

Note: It is the standard semantics in data integration.

Certain Answers Semantics



certain(q,I) = $\bigcap$ { q(J): J is a solution for I }.

Computing the Certain Answers

Theorem (FKMP): Schema mapping **M** = (**S**, **T**, Σ_{st}, Σ_t) such that:

- Σ_{st} is a set of source-to-target tgds, and
- Σ_t is the union of a weakly acyclic set of tgds with a set of egds.

Let q be a union of conjunctive queries over **T**.

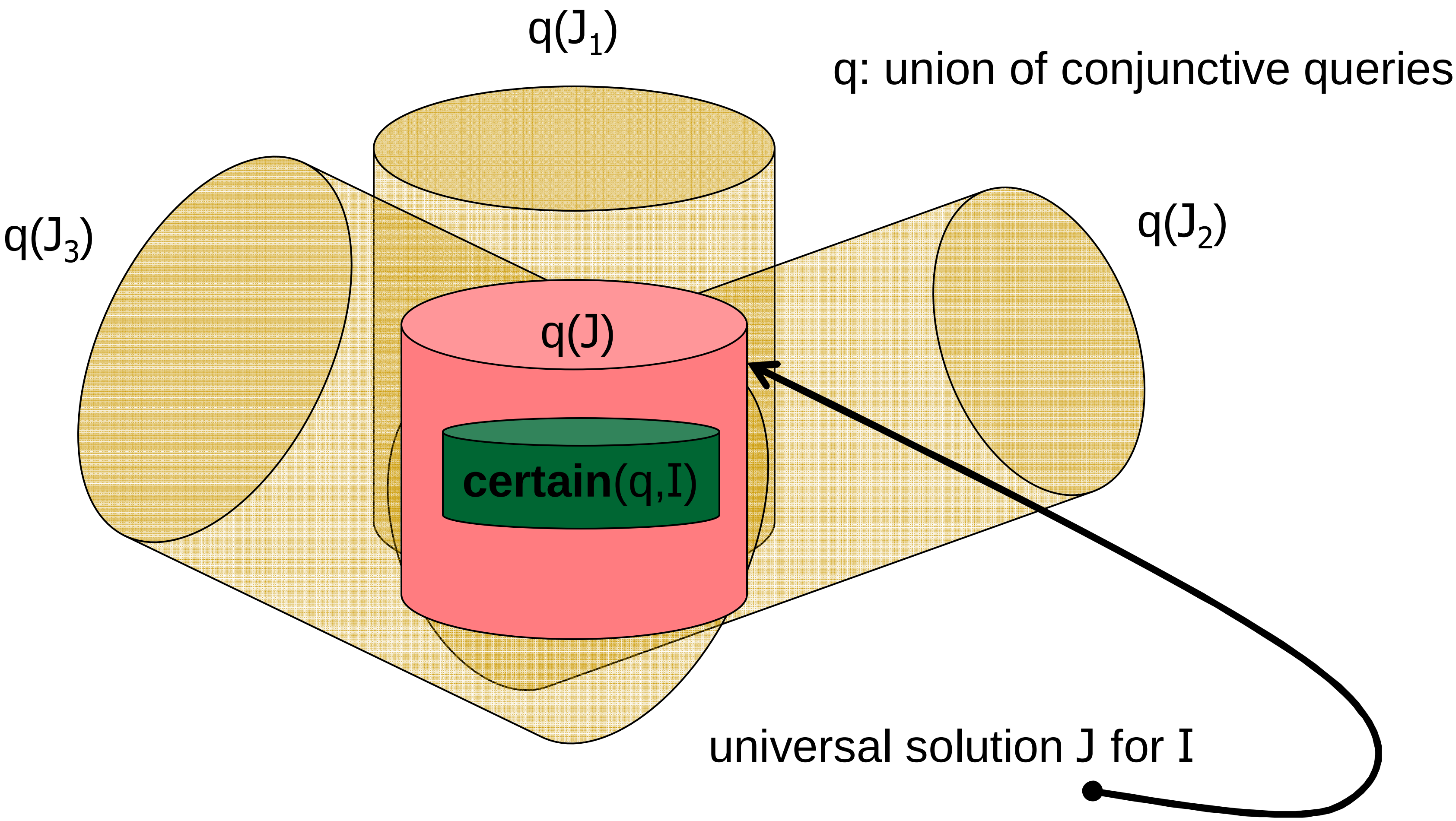
- If I is a source instance and J is a universal solution for I, then

  certain(q,I) = the set of all "null-free" tuples in q(J).

- Hence, **certain**(q,I) is computable in time polynomial in |I|:
  1. Compute a canonical universal J solution in polynomial time;
  2. Evaluate q(J) and remove tuples with nulls.

Note: This is a data complexity result (**M** and q are fixed).

Certain Answers via Universal Solutions



certain(q,I) = set of null-free tuples of q(J).

Computing the Certain Answers

Theorem (FKMP): Schema mapping **M** = (**S**, **T**, Σ_{st}, Σ_t) such that:

- Σ_{st} is a set of source-to-target tgds, and
- Σ_t is the union of a weakly acyclic set of tgds with a set of egds.

Let q be a union of conjunctive queries with inequalities ($\neq$).

- If q has at most one inequality per conjunct, then **certain**(q,I) is computable in time polynomial in |I| using a disjunctive chase.

- If q is has at most two inequalities per conjunct, then **certain**(q,I) can be coNP-complete, even if $\Sigma_t = \emptyset$.

Alternative Semantics for Query Answering

Open-World Assumption Semantics

- **certain**(q,I) = ∩ { q(J): J is a solution for I } (FKMP)
  The possible worlds for I are the solutions for I.
- **ucertain**(q,I) = ∩ { q(J): J is a universal solution for I } (FKP)
  The possible worlds for I are the universal solutions for I.

Closed-World Assumption Semantics

- Libkin 2006: CWA-Solutions
  The possible worlds for I are the members of Rep(CanSol(I)).
- Afrati and K … 2008: Semantics of aggregate queries
  The possible worlds for I are the members of End(CanSol(I)).

Closed / Open - World Assumption Semantics

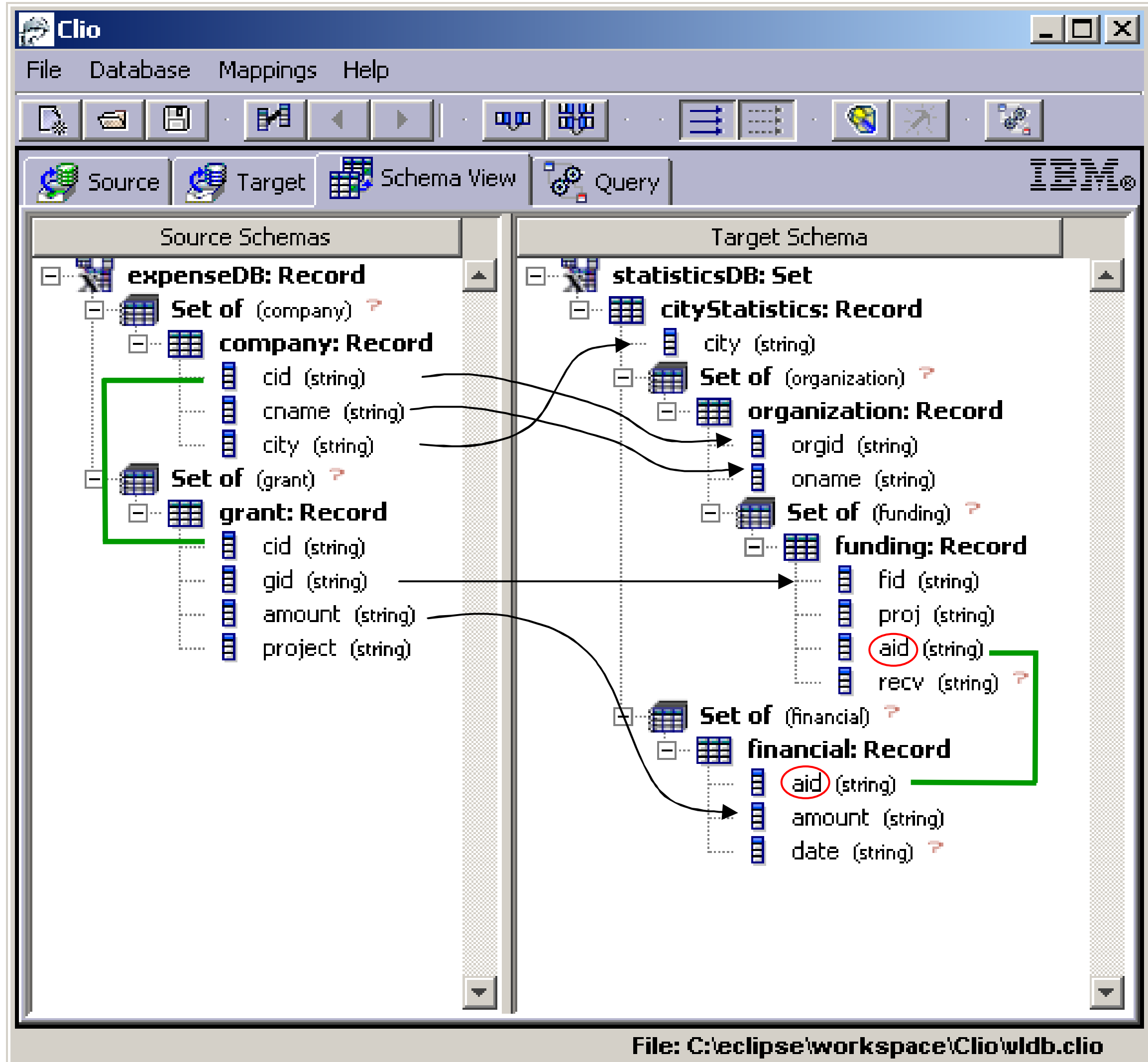
- Libkin and Sirangelo 2008

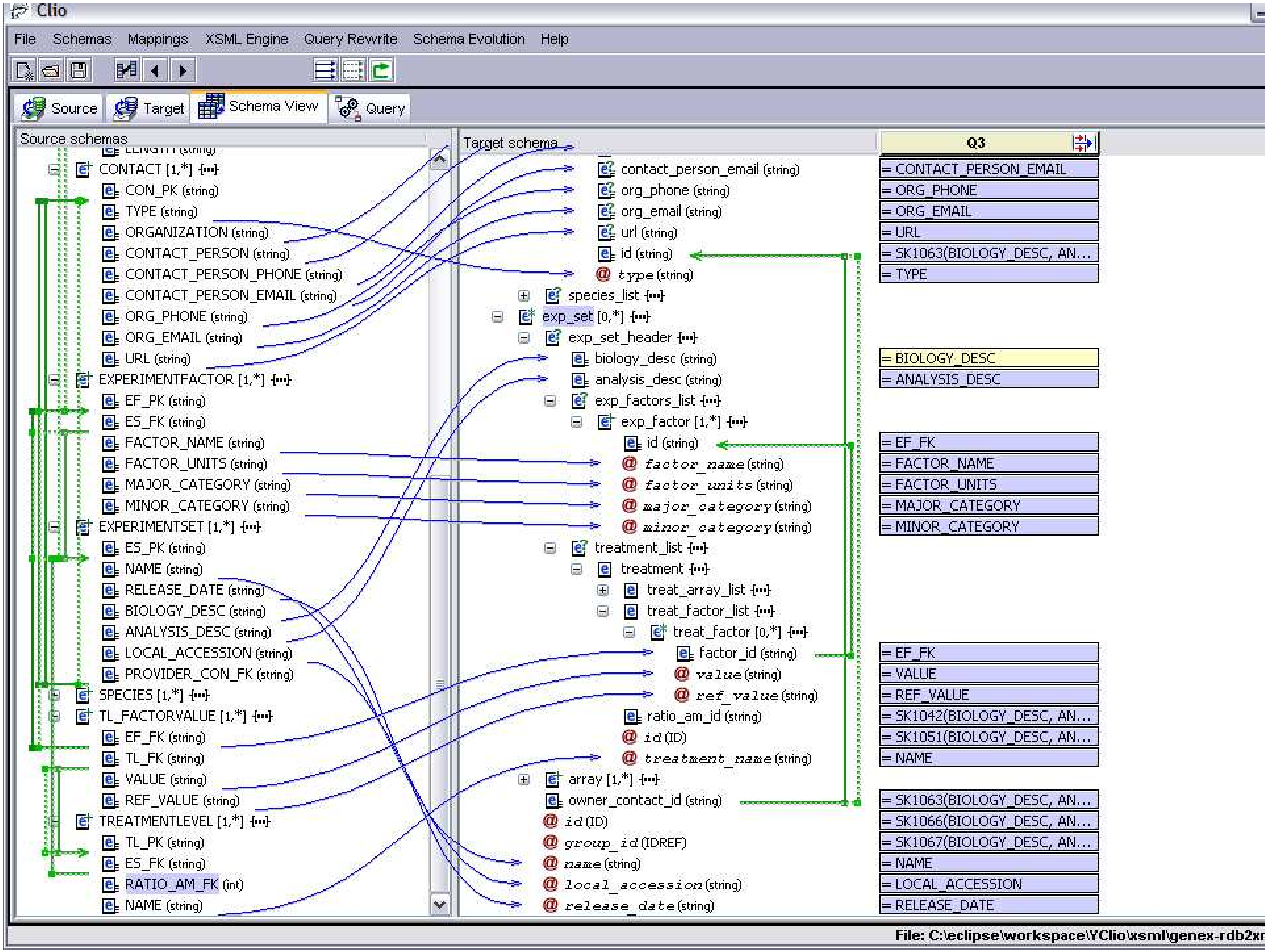
From Theory to Practice

- Clio Project at IBM Almaden managed by Howard Ho.
  - 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), provided there are no target constraints.
- Clio technology is now part of IBM Rational® Data Architect.

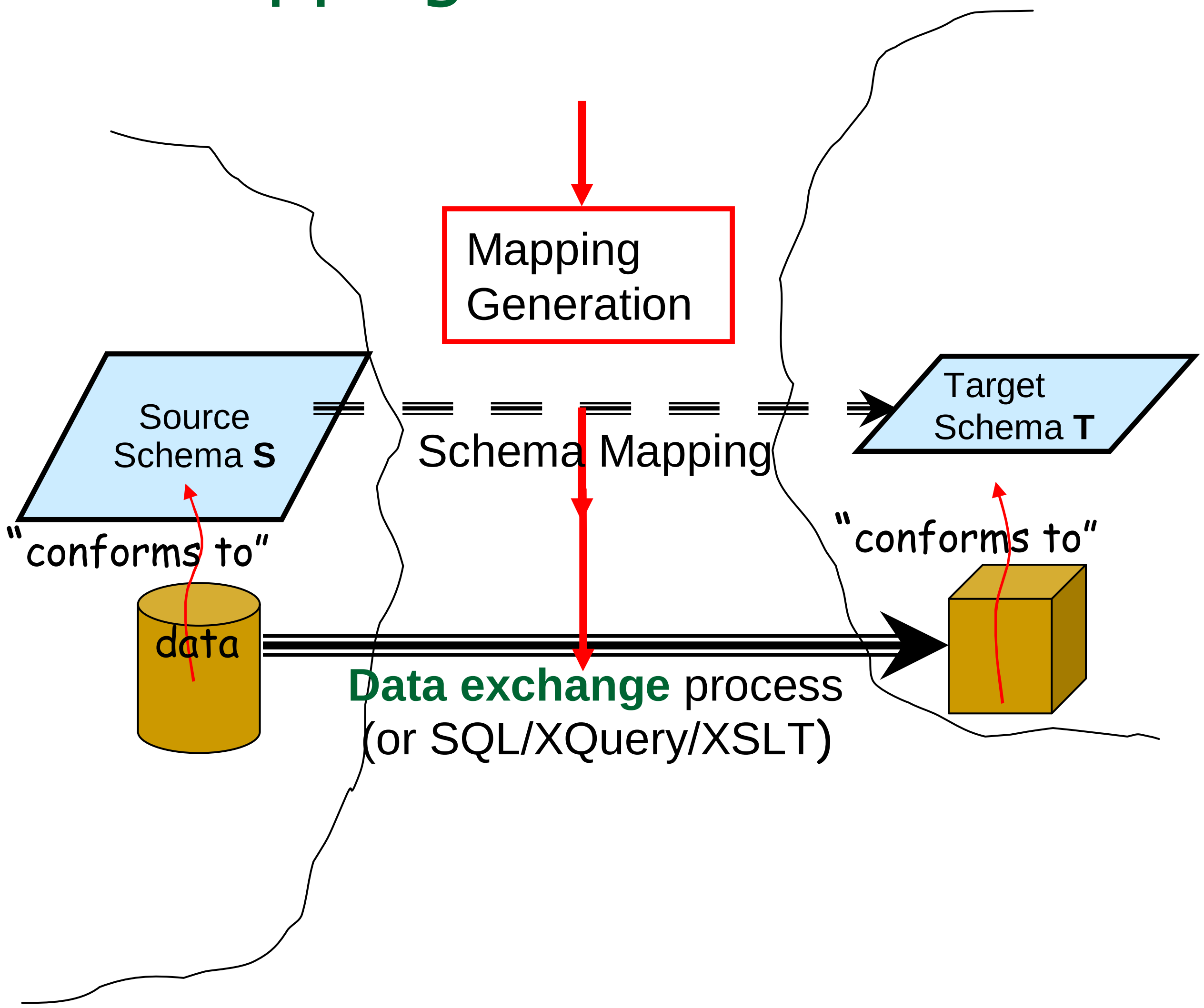
Some Features of Clio

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




Clio
File
Schemas
Mappings
XSML Engine
Query Rewrite
Schema Evolution
Help
Source
Target
Schema View
Query
Source schemas
CONTACT [1,*]
CON_PK (string)
TYPE (string)
ORGANIZATION (string)
CONTACT_PERSON (string)
CONTACT_PERSON_PHONE (string)
CONTACT_PERSON_EMAIL (string)
ORG_PHONE (string)
ORG_EMAIL (string)
URL (string)
EXPERIMENTFACTOR [1,*]
EF_PK (string)
ES_FK (string)
FACTOR_NAME (string)
FACTOR_UNITS (string)
MAJOR_CATEGORY (string)
MINOR_CATEGORY (string)
EXPERIMENTSET [1,*]
ES_PK (string)
NAME (string)
RELEASE_DATE (string)
BIOLOGY_DESC (string)
ANALYSIS_DESC (string)
LOCAL_ACCESSION (string)
PROVIDER_CON_FK (string)
SPECIES [1,*]
TL_FACTORVALUE [1,*]
EF_FK (string)
TL_FK (string)
VALUE (string)
REF_VALUE (string)
TREATMENTLEVEL [1,*]
TL_PK (string)
ES_FK (string)
RATIO_AM_FK (int)
NAME (string)
Target schema
contact_person_email (string)
org_phone (string)
org_email (string)
url (string)
id (string)
@ type (string)
species_list
exp_set [0,*]
exp_set_header
biology_desc (string)
analysis_desc (string)
exp_factors_list
exp_factor [1,*]
id (string)
@ factor_name (string)
@ factor_units (string)
@ major_category (string)
@ minor_category (string)
treatment_list
treatment
treat_array_list
treat_factor_list
treat_factor [0,*]
factor_id (string)
@ value (string)
@ ref_value (string)
ratio_am_id (string)
@ id (ID)
@ treatment_name (string)
array [1,*]
owner_contact_id (string)
@ id (ID)
@ group_id (IDREF)
@ name (string)
@ local_accession (string)
@ release_date (string)
Q3
= CONTACT_PERSON_EMAIL
= ORG_PHONE
= ORG_EMAIL
= URL
= SK1063(BIOLOGY_DESC, AN...
= TYPE
= BIOLOGY_DESC
= ANALYSIS_DESC
= EF_FK
= FACTOR_NAME
= FACTOR_UNITS
= MAJOR_CATEGORY
= MINOR_CATEGORY
= EF_FK
= VALUE
= REF_VALUE
= SK1042(BIOLOGY_DESC, AN...
= SK1051(BIOLOGY_DESC, AN...
= NAME
= SK1063(BIOLOGY_DESC, AN...
= SK1066(BIOLOGY_DESC, AN...
= SK1067(BIOLOGY_DESC, AN...
= NAME
= LOCAL_ACCESSION
= RELEASE_DATE
File: C:\eclipse\workspace\YClio\xsml\genex-rdb2xr

Schema Mappings in Clio



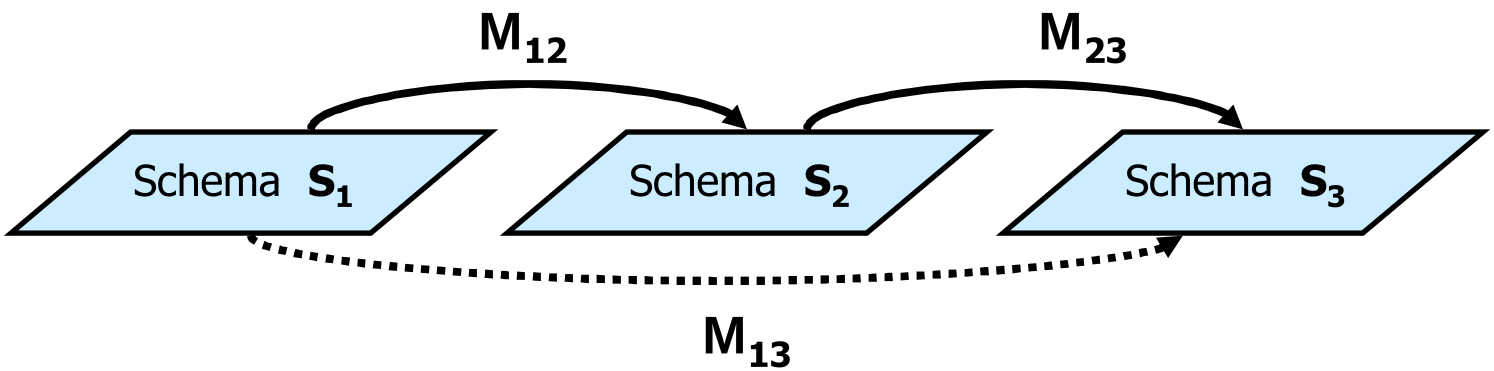
Outline of the Tutorial

- ✓ Schema Mappings and Data Exchange
- ✓ Solutions in Data Exchange
  - ✓ Universal Solutions
  - ✓ The Core of the Universal Solutions
- ✓ Query Answering in Data Exchange
- Composing and Inverting Schema Mappings

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

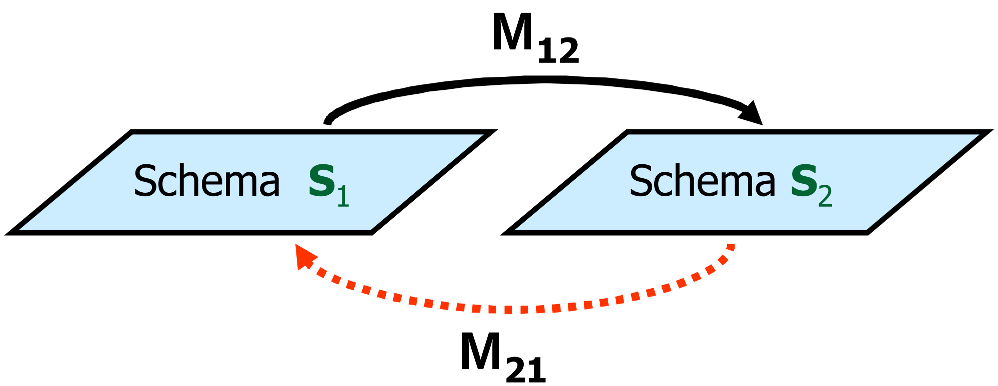
Composing Schema Mappings



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

$$\mathbf{M_{13}} = \mathbf{M_{12}} \circ \mathbf{M_{23}}$$

Inverting Schema Mapping

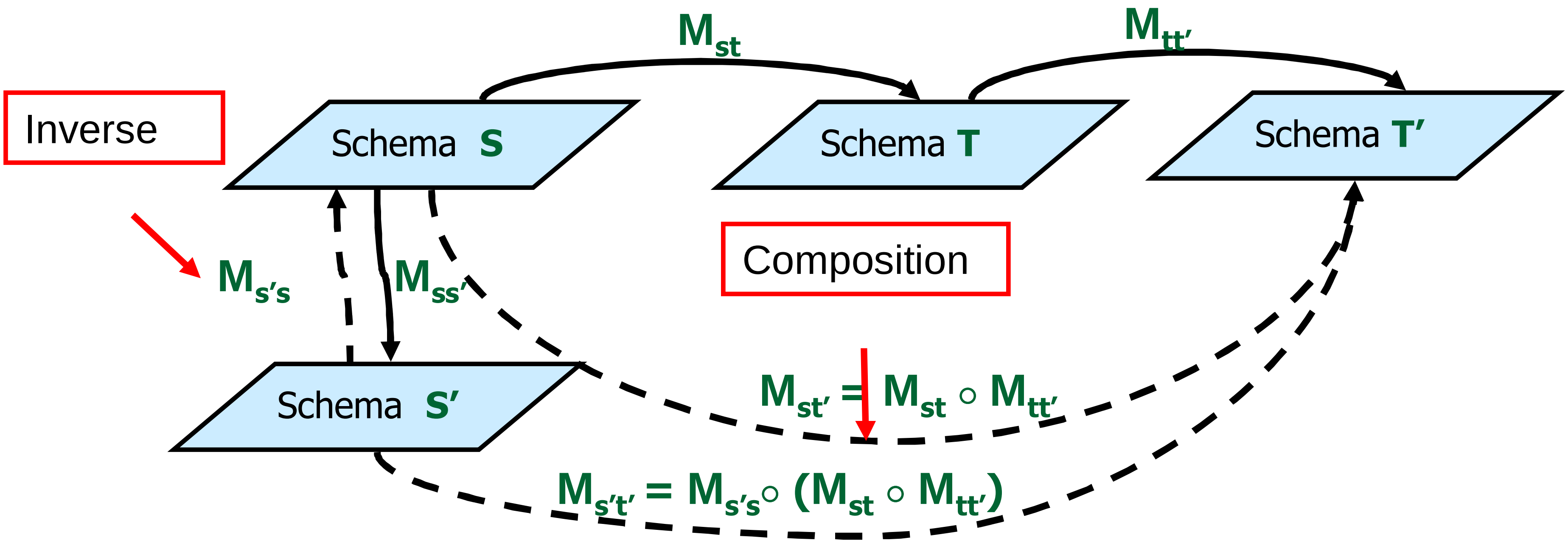


- Given $\mathbf{M_{12}}$, derive $\mathbf{M_{21}}$ that "undoes" $\mathbf{M_{12}}$

$\mathbf{M_{21}}$ is an inverse of $\mathbf{M_{12}}$

- Composition and inverse can be applied to schema evolution.

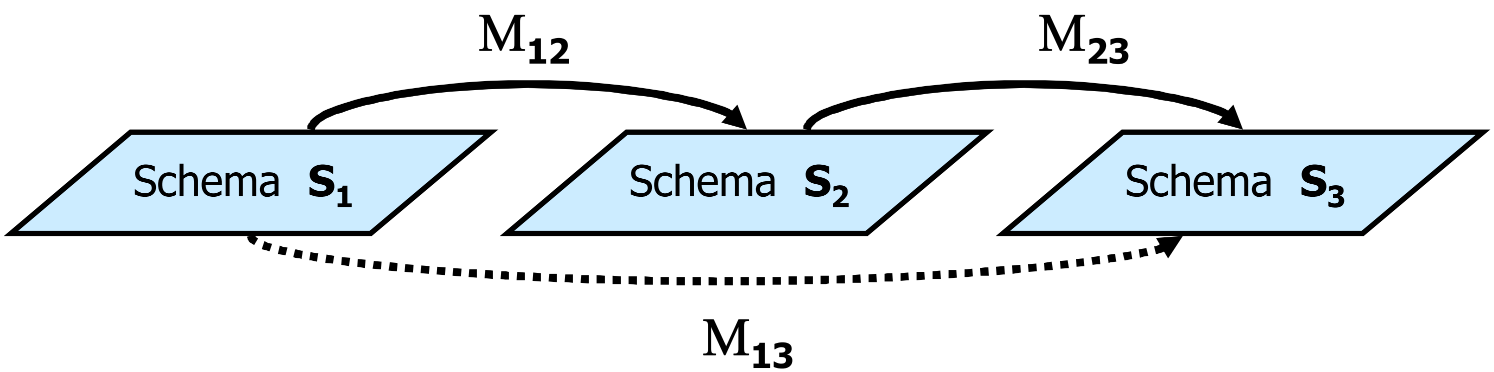
Applications to Schema Evolution



Fact:

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

Composing Schema Mappings



- Given M_{12} = ($\mathbf{S}_1$, $\mathbf{S}_2$, Σ_{12}) and M_{23} = ($\mathbf{S}_2$, $\mathbf{S}_3$, Σ_{23}), derive a schema mapping M_{13} = ($\mathbf{S}_1$, $\mathbf{S}_3$, Σ_{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 Inst($\mathbf{M}$) between instances:

$$\mathrm{Inst}(\mathbf{M}) = \{ (I,J) \mid (I,J) \models \Sigma \}.$$

- **Definition:** (FKPT 2004)

  A schema mapping $\mathbf{M}_{13}$ is a composition of $\mathbf{M}_{12}$ and $\mathbf{M}_{23}$ if

$$\mathrm{Inst}(\mathbf{M}_{13}) = \mathrm{Inst}(\mathbf{M}_{12}) \circ \mathrm{Inst}(\mathbf{M}_{23}), \text{ that is,}$$

$$(I_1,I_3) \models \Sigma_{13}$$

if and only if

there exists I_2 such that $(I_1,I_2) \models \Sigma_{12}$ and $(I_2,I_3) \models \Sigma_{23}$.

- **Note:** Also considered by S. Melnik in his Ph.D. thesis

The Composition of Schema Mappings

Fact: If both $\mathrm{M} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma)$ and $\mathrm{M'} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma')$ are compositions of M_{12} and M_{23}, then Σ are Σ' are logically equivalent. For this reason:

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

Definition: The composition query of M_{12} and M_{23} is the set

$$\mathrm{Inst}(\mathrm{M}_{12}) \circ \mathrm{Inst}(\mathrm{M}_{23})$$

(this is the model checking problem for composition)

Issues in Composition of Schema Mappings

- The semantics of composition was the first main issue.

  Some other key issues:

- 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?

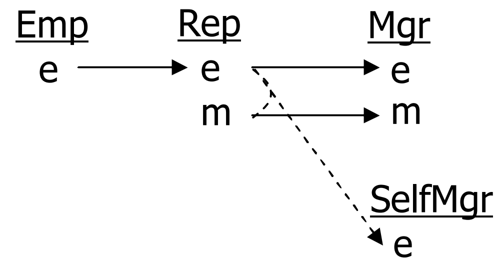
- If not, what is the "right" language for composing schema mappings?

Composition: Expressibility & Complexity

M_{12} Σ_{12}	M_{23} Σ_{23}	$M_{12} \circ M_{23}$ Σ_{13}	Composition Query
finite set of GAV (full) s-t tgds $\varphi(\mathbf{x}) \rightarrow \psi(\mathbf{x})$	finite set of s-t tgds $\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})$	finite set of s-t tgds $\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$	in PTIME
finite set of s-t tgds $\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})$	finite set of (full) s-t tgds $\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})$	may not be definable: by any set of s-t tgds; in FO-logic; in Datalog	in NP; can be NP-complete

Employee Example

- Σ_{12} :
  - Emp(e) $\rightarrow$ $\exists$m Rep(e,m)
- Σ_{23} :
  - Rep(e,m) $\rightarrow$ Mgr(e,m)
  - Rep(e,e) $\rightarrow$ SelfMgr(e)



- **Theorem:** This composition is not definable by any finite set 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

Σ_{12} :

- $\forall e \,(\text{Emp}(e) \rightarrow \exists m \,\text{Rep}(e,m))$

Σ_{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

Σ_{13} :

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

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:

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

- 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

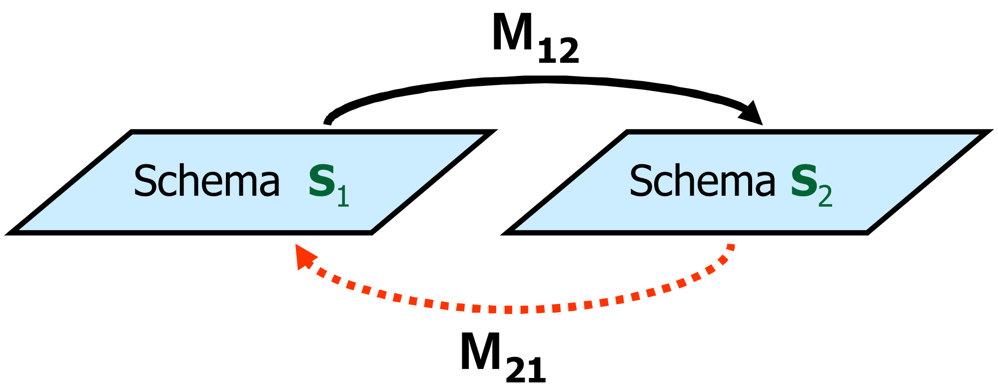
Theorem (FKPT):

- 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 finite sets of s-t tgds. Hence, SO tgds are the "right" language for the composition of 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).

Inverting Schema Mapping



- Given $\mathbf{M_{12}}$, derive $\mathbf{M_{21}}$ that "undoes" $\mathbf{M_{12}}$

$\mathbf{M_{21}}$ is an inverse of $\mathbf{M_{12}}$

- What is the "right" semantics of the inverse operator?

Approaches to Inverting Schema Mappings

- Inverses of Schema Mappings
  Fagin – PODS 2006

- Quasi-inverses of Schema Mappings
  FKPT – PODS 2007

- Maximum Recoveries of Schema Mappings
  Arenas, Pérez, Riveros -- PODS 2008

Semantics of the Inverse Operator

Fagin - 2006

- **Motivation:** an **inverse** of a function f is a function f′ s.t.

$$f \circ f' = id,$$

  where id is the **identity function** f(x)=x.

- **Idea:**
  - Define first the **identity schema mapping Id**
  - Call a schema mapping **M**′ an **inverse** of **M** if

$$\mathrm{Inst}(\mathbf{M} \circ \mathbf{M}') = \mathrm{Inst}(\mathbf{Id}).$$

Identity and Inverse

Definition: Let **S** be a schema.

Let **S*** = { R*: R $\in$ **S** }, where R* is a replica of R.

The **identity schema mapping on S** is the schema mapping

$\mathbf{Id} = (\mathbf{S}, \mathbf{S^*}, \Sigma_{\mathrm{Id}}(\mathbf{S}))$,

where $\Sigma_{\mathrm{Id}}(\mathbf{S})$ consists of the dependencies

$R(x) \rightarrow R^*(x)$, for every relation symbol R $\in$ **S**.

Definition: (Fagin) Let $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping.

A schema mapping $\mathbf{M}' = (\mathbf{T}, \mathbf{S^*}, \Sigma')$ is an **inverse** of **M** if

$$\mathrm{Inst}(\mathbf{M} \circ \mathbf{M}') = \mathrm{Inst}(\mathbf{Id}).$$

(there is J such that $(I,J) \vDash \Sigma$ and $(J,I') \vDash \Sigma'$ iff $I \subseteq I'$).

Inverting Schema Mappings

Example: Let **M** be the schema mapping specified by the tgd

$$P(x) \to Q(x,x).$$

Then:

- The schema mapping **M'** specified by the tgd

  $$Q(x,y) \to P^*(x)$$

  is an inverse of **M**.

- The schema mapping **M''** specified by the tgd

  $$Q(x,y) \to P^*(y)$$

  is also an inverse of **M**.

Note:
Inverses need not be unique up to logical equivalence.

Inverting Schema Mappings

- **Good News:**
  Rigorous semantics for the inverse operator has been given.
- **Not-so-good News:**
  It is ***rare*** that a schema mapping has an inverse.

- **Theorem:** Fagin – 2006
  If a schema mapping M has an inverse, then M must have the **unique-solutions property**:
  If I_1 and I_2 are source instances such that $\mathrm{I}_1 \neq \mathrm{I}_2$,
  then **Sol**(**M**, I_1) $\neq$ **Sol**(**M**, I_2),
  where for a source instance I, **Sol**(**M**, I) = $\{ \mathrm{J}: (\mathrm{I}, \mathrm{J}) \models \Sigma \}$.

Non-invertible Schema Mappings

Fact: **None** of the following schema mappings is invertible, as **none** satisfies the unique-solutions property:

- **Projection:**
  $P(x,y) \rightarrow Q(x)$

- **Union:**
  $P(x) \rightarrow Q(x)$
  $R(x) \rightarrow Q(x)$

- **Decomposition:**
  $P(x,y,z) \rightarrow Q(x,y) \wedge T(y,z)$

Coping with Non-invertibility

- **Difficulty:**
  - It is rare that a schema mapping is invertible.
  - The notion of an inverse is too restrictive to be useful in schema-mapping management.

- **Coping with non-invertibility** (FKPT – 2007):
  Introduce the notion of a **quasi-inverse** of a schema mapping.
  - This is a relaxation of the notion of an inverse.
  - Many non-invertible schema mappings turn out to have "natural" quasi-inverses.
  - Quasi-inverses are shown to be useful in data exchange.

Quasi-inverses: Intuition

- **Question:** How can we relax the notion of an inverse in a principled way?
- **Key Idea:** Relax the defining equation

  Inst($\mathbf{M} \circ \mathbf{M}'$) = Inst(**Id**).

  by not differentiating between instances that are

  equivalent for data-exchange purposes.
- Schema mapping $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$

  Equivalence relation $\sim_{\mathbf{M}}$ on **S**-instances:
  - $I \sim_{\mathbf{M}} I'$ if $\mathbf{Sol}(\mathbf{M}, I) = \mathbf{Sol}(\mathbf{M}, I')$.

    (i.e., for all T-instances J, we have that $(I,J) \models \Sigma$ if and only if $(I',J) \models \Sigma$.)

Quasi-inverses: Definition

Definition: Schema mapping $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$

A schema mapping $\mathbf{M}' = (\mathbf{T}, \mathbf{S^*}, \Sigma')$ is an **inverse** of $\mathbf{M}$ if

$$\mathrm{Inst}(\mathbf{M} \circ \mathbf{M}') = \mathrm{Inst}(\mathbf{Id})$$

Definition: Schema mapping $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$

A schema mapping $\mathbf{M}' = (\mathbf{T}, \mathbf{S^*}, \Sigma')$ is a **quasi-inverse** of $\mathbf{M}$ if

$$\sim_{\mathbf{M}} \circ \mathrm{Inst}(\mathbf{M} \circ \mathbf{M}') \circ \sim_{\mathbf{M}} \; = \; \sim_{\mathbf{M}} \circ \mathrm{Inst}(\mathbf{Id}) \circ \sim_{\mathbf{M}}$$

(intuitively, $\mathrm{Inst}(\mathbf{M} \circ \mathbf{M}') = \mathrm{Inst}(\mathbf{Id})$ **modulo** $\sim_{\mathbf{M}}$).

Quasi-inverse: Definition

Definition: Schema mapping $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$

A schema mapping $\mathbf{M}' = (\mathbf{T}, \mathbf{S^*}, \Sigma')$ is a **quasi-inverse** of $\mathbf{M}$ if

$$\sim_{\mathbf{M}} \circ \text{Inst}(\mathbf{M} \circ \mathbf{M}') \circ \sim_{\mathbf{M}} \; = \; \sim_{\mathbf{M}} \circ \text{Inst}(\mathbf{Id}) \circ \sim_{\mathbf{M}}$$

This means that the following are equivalent for I_1 and I_2:

1) There are I_1', I_2', and J such
   - $I_1 \sim_{\mathbf{M}} I_1'$, $I_2 \sim_{\mathbf{M}} I_2'$ and
   - $(I_1', J) \models \Sigma$ and $(J, I_2') \models \Sigma'$.
2) There are I_1'' and I_2'' such that
   - $I_1 \sim_{\mathbf{M}} I_1''$, $I_2 \sim_{\mathbf{M}} I_2''$ and
   - $I_1'' \subseteq I_2''$.

Quasi-inverses of Schema Mappings

Summary of main results:

- **Exact structural criterion** for the existence of quasi-inverses.
- **Complete characterization** of the language needed to express quasi-inverses, if they exist.
- **Algorithm** for producing a "good" quasi-inverse, if one exists.
- **Applications** of quasi-inverses to data exchange.

Criterion for the Existence of Quasi-inverses

Theorem : Let **M** = (**S**, **T**, Σ) be a schema mapping in which Σ is a set of s-t tgds. The following statements are equivalent:

- **M** has a quasi-inverse.
- **M** has the **subset-property:**
  For every pair (I_1, I_2) of **S**-instances such that
  Sol(**M**, I_2) $\subseteq$ **Sol**(**M**, I_1),
  there is a pair (I_1', I_2') of **S**-instances such that
  - $I_1 \sim_{\mathbf{M}} I_1'$ and $I_2 \sim_{\mathbf{M}} I_2'$
  - $I_1' \subseteq I_2'$.

Applications of the Subset Property

Proposition:

- Every LAV mapping has a quasi-inverse.
- There is a GAV mapping that has no quasi-inverse.

Proof:

- Show that every LAV mapping has the **subset property**.
- Show that the GAV mapping specified by the tgds

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

  $E(x,z) \wedge E(z,y) \rightarrow M(z)$

  does **not** have the **subset property**.

Quasi-invertible, Non-invertible Mappings

- **Projection:** $P(x,y) \rightarrow Q(x)$
  Quasi-inverse: $Q(x) \rightarrow \exists\, y\ P(x,y)$

- **Union:** $P(x) \rightarrow Q(x)$
  $R(x) \rightarrow Q(x)$
  Quasi-inverse #1: $Q(x) \rightarrow P(x) \vee R(x)$
  Quasi-inverse #2: $Q(x) \rightarrow P(x)$
  Quasi-inverse #3: $Q(x) \rightarrow R(x)$

- **Decomposition:** $P(x,y,z) \rightarrow Q(x,y) \wedge T(y,z)$
  Quasi-inverse #1: $Q(x,y) \wedge T(y,z) \rightarrow P(x,y,z)$
  Quasi-inverse #2: $Q(x,y) \rightarrow \exists\, z\ P(x,y,z)$
  $T(y,z) \rightarrow \exists\, x\ P(x,y,z)$

The Language of Quasi-inverses

Theorem: Let **M** = (**S**, **T**, Σ) be a schema mapping in which Σ is a set of s-t tgds. Assume that **M** has a quasi-inverse. Then:

- **M** has a quasi-inverse **M′** specified by a set of **disjunctive tgds with constants and inequalities.**
- There is an (exponential) algorithm QI for producing such an **M′**.
- **No** smaller language can express quasi-inverses.

Disjunctive tgds with constants and inequalities:

$$\phi(\mathbf{x}) \rightarrow \vee_i \exists \mathbf{y}_i \psi_i(\mathbf{x}, \mathbf{y}_i),$$ where

- $\phi(\mathbf{x})$ is a conjunction of
  - **T**-atoms;
  - Formulas of the form Constant(x), where x is a variable in $\mathbf{x}$;
  - Inequalities $x \neq x'$, where x, x′ are variables in $\mathbf{x}$.
- Each $\psi_i(\mathbf{x}, \mathbf{y}_i)$ is a conjunction of **S**-atoms.

The Language of Quasi-inverses

Theorem: Every LAV schema mapping has a quasi-inverse specified by a set of **tgds with constants and inequalities**. Thus, **disjunctions** are **not** needed.

Theorem: Every quasi-invertible GAV schema mapping has a quasi-inverse specified by a set of **disjunctive tgds with inequalities**. Thus, **constants** are **not** needed.

Necessity of the Language – Sample Results

Necessity of Constants:

LAV Schema Mapping M: $E(x,y) \rightarrow \exists z\, (F(x,z) \wedge F(z,y))$

- Quasi-inverse M':
  $F(x,z) \wedge F(z,y) \wedge Constant(x) \wedge Constant(y) \rightarrow E(x,y)$
- M has **no** quasi-inverse specified by **disjunctive tgds with inequalities**.

Necessity of Disjunctions:

GAV Schema Mapping M:

$P_1(x) \rightarrow S_1(x)$, $P_2(x) \rightarrow S_1(x)$

$P_3(x) \rightarrow S_2(x)$, $P_4(x) \rightarrow S_2(x)$

$P_i(x) \wedge P_j(x) \rightarrow R_{ij}(x)$, $i = 1,2$ and $j = 3,4$

- M has a quasi-inverse
- M has **no** quasi-inverse specified by **tgds with constants and inequalities**.

The Language of Quasi-inverses: Summary

	Inequalities needed? $x \neq x'$	**Constants needed?** Constant(x)	**Disjunctions needed?** $\vee_i \exists \psi_i(\mathbf{x},\mathbf{y}_i)$
LAV Mappings	**Yes**	**Yes**	**No**
GAV Mappings	**Yes**	**No**	**Yes**
Mappings specified by arbitrary tgds	**Yes**	**Yes**	**Yes**

Quasi-inverses in Data Exchange

Theorem: Let **M** be a quasi-invertible schema mapping and let **M'** be the schema mapping produced by the QI-algorithm. Then **M'** can be used to produce source instances that are "**data exchange equivalent**" to a given source instance.

More formally, for every **S**-instance I, we have that

$$\mathrm{chase}_{\mathbf{M}}(\mathrm{chase}_{\mathbf{M'}}(\mathrm{chase}_{\mathbf{M}}(I)))$$

contains a **T**-instance J that is **homomorphically equivalent** to $\mathrm{chase}_{\mathbf{M}}(I)$.

Synopsis of Results about Quasi-inverses

- Quasi-inverses are a useful relaxation of inverses.
- Exact combinatorial criterion for the existence of quasi-inverses.
- Complete characterization of the language of quasi-inverses.
- Quasi-inverses are useful in data exchange.

Maximum Recoveries of Schema Mappings

Definition (Arenas, Pérez, Riveros – 2008):

M = (**S**, **T**, Σ) a schema mapping

- **M'** = (**T**, **S**, Σ') is a recovery of **M** if for every source instance I, we have that (I,I) $\in$ Inst(**M**$\circ$**M'**).
- **M'** = (**T**, **S**, Σ') is a maximum recovery of **M** if it is a recovery and there is no recovery **M''** of **M** such that

  Inst(**M**$\circ$**M''**) $\subset$ Inst(**M**$\circ$**M'**).

Summary of Main Results

- Exact criterion for the existence of maximum recoveries.
- Characterization of the language for expressing maximum recoveries of schema mappings specified by s-t tgds.
- Algorithm for constructing a maximum recovery of a schema mapping specified by s-t tgds.

Maximum Recoveries vs. Quasi-inverses

Theorem (Arenas, Pérez, Riveros – 2008):

M = (**S**, **T**, Σ) a schema mapping specified by s-t tgds.

- **M** has a maximum recovery.
- If **M** is invertible, then **M'** is a maximum recovery of **M** if and only if **M'** is an inverse of **M**.
- If **M** is quasi-invertible, then **M** is a maximum recovery of **M** if and only if **M'** is both a recovery and a quasi-inverse of **M**.

Some Directions of Research

- Inverting schema mapings requires further study.
- Detailed study of other schema mapping operators (Diff, Merge, …) remains to be carried out.
- Applications of schema-mapping operators to:
  - Study of schema evolution;
  - Modeling and analysis of ETL via schema mappings.

Related Work (very partial list)

- XML Data Exchange
  (Arenas and Libkin – 2005).

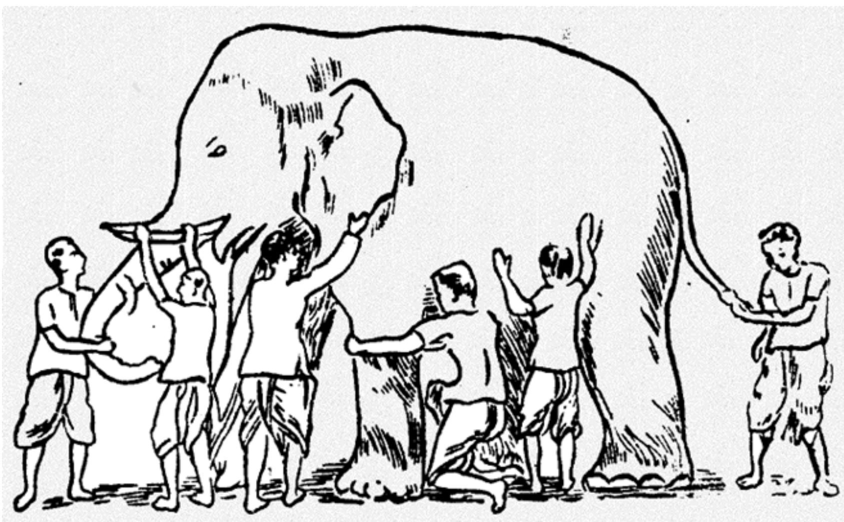
- Schema mappings with arithmetic comparisons
  (Afrati, Li, Pavlaki – 2008).

- Composing richer schema mappings
  (Nash, Bernstein, Melnik – 2007)

- Peer data exchange
  (Fuxman, K …, Miller, Tan – 2007)

- Schema-mapping optimization
  (FKNP – 2008)

Data Interoperability: The Elephant and the Six Blind Men



- Data interoperability remains a major challenge:

  "Information integration is a beast." (L. Haas – 2007)

- Schema mappings specified by tgds offer a formalism that covers only some aspects of data interoperability.

- However, theory and practice can inform each other.