Logic and Databases

1st EATCS School for Young Researchers

Phokion G. Kolaitis

University of California Santa Cruz & IBM Research – Almaden

Outline

Part II: Database Dependencies and Data Exchange

- Functional and Inclusion Dependencies
- The Implication Problem for Database Dependencies
- Data-interoperability via Database Dependencies
- Schema Mappings and Data Exchange
- Managing Schema Mappings

Unifying Theme:

The use of logic as a specification language in data management

Logic and Databases

- Extensive interaction between logic and databases during the past 40 years.
- Logic provides both a unifying framework and a set of tools for formalizing and studying data management tasks.
- The interaction between logic and databases is a prime example of
  - Logic **in** Computer Science

    but also
  - Logic **from** Computer Science

Two Main Uses of Logic in Databases

- Logic as a formalism for expressing database query languages
  - Relational Calculus = First-Order Logic
  - Datalog = Existential Positive First-Order Logic + Recursion

- Logic as a specification language for expressing database dependencies, i.e., semantic restrictions (integrity constraints) that the data of interest must obey.
  - Functional dependencies
  - Inclusion dependencies.

Functional Dependencies

- Relational Schema:

  R(student, course, grade)
- Database Dependency:

  Every student enrolled in a course is assigned a unique grade
- Expressed as a functional dependency
  - student, course $\rightarrow$ grade
- Expressed in first-order logic
  - $\forall$ s, c, g, g' (R(s,c,g) $\wedge$ R(s,c,g') $\rightarrow$ g = g')
- Special case of an equality-generating dependency
  - $\forall$ **x** (φ(**x**) $\rightarrow$ $x_i = x_j$), where

    φ(**x**) is a conjunction of atomic formulas.

Inclusion Dependencies

- Relational Schemas:

  R(student, course, grade) and T(course, teacher)
- Database Dependency:

  For every triple in R there is a pair in T with the same value for course.
- Expressed as an inclusion dependency
  - R[course] $\subseteq$ T[course]
- Expressed in first-order logic
  - $\forall$ s, c, g(R(s,c,g) $\rightarrow$ $\exists$ t T(c,t))
- Special case of a tuple-generating dependency
  - $\forall$ **x** (φ(**x**)$\rightarrow$ $\exists$ **y** ψ(**x**, **y**)), where

    φ(**x**), ψ(**y**) are conjunctions of atomic formulas.

Database Dependencies

- Numerous different classes of database dependencies were introduced and studied in the 1970s and the 1980s.

- They all turned out to be expressible in first-order logic.
  In fact, they turned out to be expressible in one of the following two fragments of first-order logic:
  - equality-generating dependencies
  - tuple-generating dependencies.

- Main focus of the study of database dependencies:
  The Implication Problem for database dependencies.

The Implication Problem

- Definition: Σ a set of dependencies and θ a dependency.
  Σ logically implies θ, denoted $\Sigma \models \theta$, if for every database D satisfying every dependency in Σ, we have that D satisfies θ.

- Definition: C a class of database dependencies.
  The implication problem for C is the decision problem:
  Given a finite set Σ of dependencies from C and a dependency θ in C, does $\Sigma \models \theta$?

The Implication Problem

- Clearly, if C is the class of all FO-sentences, then the implication problem for C is undecidable.
- There are, however, natural classes of database dependencies that are expressible in FO and whose implication problem is decidable.
- Here, we will focus on the implication problem for the class of functional dependencies and the class of inclusion dependencies.

Functional Dependencies

Definition: R be a relational schema.

- An instance r of R satisfies the functional dependency

$$A_1,\ldots,A_m \rightarrow B_1,\ldots,B_k$$

if there are no two tuples in R that have the same value on the attributes $A_1,\ldots,A_m$, but differ on at least one of the values of $B_1,\ldots,B_k$.

  - In other words, the values of the attributes $B_1,\ldots,B_k$ are a function of the values of the attributes $A_1,\ldots,A_m$.

- We say that R satisfies the functional dependency

$$A_1,\ldots,A_m \rightarrow B_1,\ldots,B_k$$

if every instance r of R satisfies $A_1,\ldots,A_m \rightarrow B_1,\ldots,B_k$.

  - This is a semantic restriction imposed on all “legal” instances of the relational schema R.

Functional Dependencies

Question: How do we know that a FD holds on a relational schema?

Answer:

- This is semantic information that is provided by the customer who wishes to have a database designed for the data of interest.

- A FD may be derived (inferred) from other known FDs about the schema. This is what the implication problem is all about.

Functional Dependencies

Example: COMPANY(employee, dpt, manager)

- Some plausible FDs are:
  - employee $\rightarrow$ dpt
  - dpt $\rightarrow$ manager
  - manager $\rightarrow$ dpt
  - employee $\rightarrow$ manager

  Each models a different aspect of the data at hand
- Some implausible FDs are:
  - manager $\rightarrow$ employee
  - dpt $\rightarrow$ employee

- Note: If both employee $\rightarrow$ dpt and dpt $\rightarrow$ manager hold, then employee $\rightarrow$ manager must also hold. This is an example of logical implication of a functional dependency from given ones.

Reasoning about Functional Dependencies

Definition: Assume that R is a relational schema, F is a set of FDs, and $X \rightarrow Y$ is a FD (all with attributes from R).
We say that F logically implies $X \rightarrow Y$ (and write $F \models X \rightarrow Y$),
if for every instance r of R that satisfies F, we have that r also satisfies $X \rightarrow Y$.

Examples:

- Transitivity Rule: $\{A \rightarrow B, B \rightarrow C\} \models A \rightarrow C$.
- Augmentation Rule: $\{A \rightarrow B\} \models AC \rightarrow BC$, for every attribute C.

The Implication Problem for Functional Dependencies

Problem 1: Given a relational schema R, a set F of FDs on R, and a FD $X \rightarrow Y$, determine whether or not $F \models X \rightarrow Y$.

Question: What is the computational complexity of this problem?

Theorem (Beeri and Bernstein – 1979):
The implication problem for the class FD of functional dependencies is in PTIME. In fact, it is solvable in linear time.

The Implication Problem for Functional Dependencies

Problem 1: Given a relational schema R, a set F of FDs on R, and a FD $X \rightarrow Y$, determine whether or not $F \models X \rightarrow Y$.

Fact: The following are equivalent:

- $F \models X \rightarrow Y$, where $Y = \{B_1,\ldots,B_k\}$
- $F \models X \rightarrow B_i$, for every $B_i \in Y$.

Definition: $X^+ = \{B: F \models X \rightarrow B\}$ is the closure of X with respect to F.

Fact: The following statements are equivalent:

- $F \models X \rightarrow Y$
- $Y \subseteq X^+$.

Algorithmic Problems about Functional Dependencies

Problem 1: Given a relational schema R, a set F of FDs on R, and a FD $X \rightarrow Y$, determine whether or not $F \models X \rightarrow Y$.

Fact: The following statements are equivalent:

- $F \models X \rightarrow Y$
- $Y \subseteq X^+$.

Consequently, to solve Problem 1, it suffices to solve Problem 2 below.

Problem 2: Given a relational schema R, a set F of FDs on R, and a set X of attributes, compute X^+.

The Closure Algorithm

Input: Relational schema R with attribute set U, set F of FDs, $X \subseteq U$

Output: $X^+ = \{B: F \models X \rightarrow B\}$.

Initialization Step: $X_0 = X$

Recursive Step:

$X_{n+1} = X_n \cup \{B$: there is a FD $Y \rightarrow Z$ in F such that $Y \subseteq X_n$ and $B \in Z\}$

Stopping Rule: When $X_n = X_{n+1}$ for the first time, stop and output X_n.

Closure Algorithm

Example:

- R(A,B,C,D,E,H,G)
- $F = \{ AB \rightarrow CD, C \rightarrow EH, D \rightarrow G \}$

Compute $\{A,C\}^+$ and $\{A,B\}^+$

- $X = \{A,C\}$
  - $X_0 = \{A,C\}$
  - $X_1 = \{A,C\} \cup \{E,H\} = \{A,C,E,H\}$
  - $X_2 = X_1.$
  - Hence, $\{A,C\}^+ = \{A,C,E,H\}$, which implies that {A,C} is not a superkey.
- $X = \{A,B\}$
  - $X_0 = \{A,B\}$
  - $X_1 = \{A,B\} \cup \{C,D\} = \{A,B,C,D\}$
  - $X_2 = \{A,B,C,D\} \cup \{E,H,G\} = \{A,B,C,D,E,H,G\}$
  - $X_3 = X_2$
  - Hence, $\{A,B\}^+ = \{A,B,C,D,E,H,G\}$, which implies that {A,B} is a superkey.

Properties of the Closure Algorithm

Input: Relational schema R with attribute set U, set F of FDs, $X \subseteq U$

Output: $X^+ = \{B:\ F \models X \rightarrow B\}$.

Initialization Step: $X_0 = X$

Recursive Step:

$X_{n+1} = X_n \cup \{B$: there is a FD $Y \rightarrow Z$ in F such that $Y \subseteq X_n$ and $B \in Z\}$

Stopping Rule: When $X_n = X_{n+1}$ for the first time, stop and output X_n.

Facts:

- Termination: The closure algorithm terminates within at most |U| iterations.
- Correctness: The closure algorithm outputs X^+.
  - Soundness: If B is in the output of the algorithm, then $X \rightarrow B$.
  - Completeness: If $X \rightarrow B$, then B is in the output of the algorithm.

Properties of the Closure Algorithm

- Termination: The closure algorithm terminates within |U| iterations.
  Proof: $X=X_0 \subseteq X_1 \subseteq \ldots \subseteq X_n \subseteq X_{n+1} \subseteq \ldots \subseteq U$.

- Correctness: Let W be the output of the algorithm on input X.
  Show that $W = X^+$. This breaks down to two different tasks.
- Soundness: $W \subseteq X^+$
  Proof: By induction on n, show that $X_n \subseteq X^+$, for all n.
  - Base Step: $X_0 = X \subseteq X^+$.
  - Inductive Step: Assume that $X_n \subseteq X^+$. Show that $X_{n+1} \subseteq X^+$.
    (Exercise).
- Completeness: $X^+ \subseteq W$.
  This requires some work.

Properties of the Closure Algorithm

- Completeness: $X^+ \subseteq W$.
  - We know that $X \subseteq W$. Hence, $X^+ \subseteq W^+$ (Why?).
  - So, it suffices to show that $W^+ \subseteq W$. This means that if $F \models W \rightarrow B$, then $B \in W$ or, equivalently, that

    if B is not W, then F does not logically imply $W \rightarrow B$.
  - So, let B be an attribute of R such that B is not in W.
  - Construct a relation r consisting of two tuples s and t such that
    - $s(A) = t(A)$, if A is in W.
    - $s(A) \neq t(A)$, if A is not in W. In particular, $s(B) \neq t(B)$.
  - By construction, we have that r does not satisfy $W \rightarrow B$.
  - On the other hand, it is easy to see that r satisfies every FD in F (exercise). Hence, F does not logically imply $W \rightarrow B$.

The Closure Algorithm: Summary

- The running time of the closure algorithm is quadratic in the size (length) of F and X (why?).
- The closure algorithm can be refined to run in linear time in the size of F and X.
- The closure algorithm can be used to determine whether $F \models X \rightarrow Y$ (by testing that $Y \subseteq X^+$).
- Hence, the implication problem for FDs is solvable in linear time.
- By applying the closure algorithm repeatedly, we can compute all superkeys and candidate keys (exponential-time algorithm).

Reasoning about Functional Dependencies via Rules

- In 1974, W.W. Armstrong suggested a set of rules for reasoning about functional dependencies.
- These rules became known as **Armstrong's Axioms**.

In what follows, X, Y, Z stand for sets of attributes of a relation schema R.

Armstrong's Axioms

- **A1. Reflexivity:**
- If $X \subseteq Y$, then $Y \rightarrow X$ is an axiom *(trivial dependencies*).
- **A2. Augmentation:**

  From $X \rightarrow Y$, infer $XZ \rightarrow YZ$, where Z is an arbitrary set of attributes.
- **A3. Transitivity:**

  From $X \rightarrow Y$ and $Y \rightarrow Z$, infer $X \rightarrow Z$.

Reasoning with Armstrong's Axioms

Definition: Consider relation schema R, set F of FDs, and FD $X \rightarrow Y$.
We say that F infers $X \rightarrow Y$, denoted $F \vdash X \rightarrow Y$, if

- $X \rightarrow Y \in F$
- $X \rightarrow Y$ is a Reflexivity Axiom (A1).
- $X \rightarrow Y$ can be inferred from previously inferred functional dependencies using Augmentation (A2) or Transitivity (A3).

In other words, $F \vdash X \rightarrow Y$ if there is a sequence,,

$$X_1 \rightarrow Y_1, X_2 \rightarrow Y_2, \ldots, X_n \rightarrow Y_n \,,$$

called a derivation from F such that

- $X_n \rightarrow Y_n = X \rightarrow Y$ and for each i with $1 \leq i \leq n$:
- $X_i \rightarrow Y_i$ is in F, or
- $X_i \rightarrow Y_i$ is a Reflexivity Axiom (A1), or
- $X_i \rightarrow Y_i$ follows from earlier members of the sequence using Augmentation (A2) or Transitivity (A3).

Reasoning with Armstong's Axioms

Example: Show that

$$\{AB \rightarrow CD,\ C \rightarrow EH,\ D \rightarrow G\} \vdash AB \rightarrow H$$

Derivation:

1. $AB \rightarrow CD$ (in F)
2. $CD \rightarrow C$ (A1)
3. $AB \rightarrow C$ (A3 on 1. and 2.)
4. $C \rightarrow EH$ (in F)
5. $AB \rightarrow EH$ (A3. on 3. and 4.)
6. $EH \rightarrow H$ (A1)
7. $AB \rightarrow H$ (A3 on 5. and 6.)

Reasoning with Armstong's Axioms

Example: Show that

$$\{AB \rightarrow CD,\ C \rightarrow EH,\ D \rightarrow G\} \vdash AB \rightarrow EHG$$

Derivation:

1. $AB \rightarrow CD$ (in F)
2. $C \rightarrow EH$ (in F))
3. $CD \rightarrow EHD$ (A2 on 2.)
4. $D \rightarrow G$ (in F)
5. $EHD \rightarrow EHG$ (A2 on 4.)
6. $CD \rightarrow EHG$ (A3 on 3. and 5.)
7. $AB \rightarrow EHG$ (A3 on 1. and 6.)

Soundness and Completeness of Amstrong's Axioms

Theorem: Let R be a relation schema, F a set of functional dependencies on R, and $X \rightarrow Y$ a functional dependency.

Then the following statements are equivalent:

1) $F \vdash X \rightarrow Y$ (syntactic notion)
2) $F \models X \rightarrow Y$ (semantic notion)

Proof: Essentially the same as the correctness of the Closure Algorithm.

Note:

1) $\Rightarrow$ 2) : Soundness Theorem (easier direction)

2) $\Rightarrow$ 1) : Completeness Theorem (harder direction)

Note: Armstrong's Theorem shows that a semantic notion coincides with a syntactic notion.

Inclusion Dependencies

Example: ENROLLS(student-id, name, course),
PERFORM(student-id, course, grade)

Consider the integrity constraint:

- "every student enrolled in a course is assigned a grade"

This is an example of an inclusion dependency;
it is denoted by:

ENROLLS[student-id,course] $\subseteq$ PERFORM[student-id,course,].

Inclusion Dependencies

Definition: An inclusion dependency (ID) is an expression of the form

$$S[A_1,\ldots,A_n] \subseteq T[B_1,\ldots,B_n], \text{ where}$$

- $A_1,\ldots,A_n$ are distinct attributes from S'
- $B_1,\ldots,B_n$ are distinct attributes from T' with data types matching those of $A_1,\ldots,A_n$

- A database instance D satisfies $S[A_1,\ldots,A_n] \subseteq T[B_1,\ldots,B_n]$ if for every tuple $s \in S$ with values $c_1,\ldots,c_n$ for the attributes $A_1,\ldots,A_n$, there is a tuple $t \in T$ with values $c_1,\ldots,c_n$ for the attributes $B_1,\ldots,B_n$.

- A database schema satisfies $S[A_1,\ldots,A_n] \subseteq T[B_1,\ldots,B_n]$ if every instance D of the schema satisfies this ID.

Inclusion Dependencies and First-Order Logic

Fact: Every inclusion dependency $S[A_1,\ldots,A_n] \subseteq T[B_1,\ldots,B_n]$ can be expressed in FO-logic.

Proof (by example): Consider the ID

ENROLLS[student-id,course] $\subseteq$ PERFORM[student-id,course,grade],

which expresses the integrity constraint:

"every student enrolled in a course is assigned a grade".

This ID is equivalent to the relational calculus formula

$$\forall x,y,z\ (\text{ENROLLS}(x,y,z) \rightarrow \exists w\ \text{PERFORM}(x,z,w)).$$

Note: Unlike functional dependencies, inclusion dependencies are integrity constraints between two (usually different) relational schemas.

The Implication Problem for Inclusion Dependencies

Theorem (Casanova, Fagin, Papadimitriou – 1984)
The implication problem for the class IND of inclusion dependencies is PSPACE-complete.

Note: $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PH} \subseteq \mathrm{PSPACE}$.

Proof Hint:

- Membership in PSPACE:

  Non-deterministic polynomial-space algorithm + Savitch's Theorem (NPSPACE = PSPACE).
- PSPACE-hardness:

  Reduction from Linear Bounded Automaton Acceptance.

The Implication Problem for FDs and INDs

Question: What can we say about the implication problem for the class FD $\cup$ IND, i.e., for the union of the class of functional dependencies with the class of inclusion dependencies?

Theorem (Mitchell – 1983, Chandra & Vardi – 1985)
The implication problem for the class FD $\cup$ IND of functional and inclusion dependencies is undecidable.

Proof Hint:
Reduction from the Word Problem for Monoids.

The Implication Problem for Database Dependencies

- FDs are a special case of equality-generating dependencies (egds)

  $\forall \mathbf{x} \ (\varphi(\mathbf{x}) \rightarrow x_i = x_j)$, where

  $\varphi(\mathbf{x})$ is a conjunction of atomic formulas.
- IND are a special case of tuple-generating dependencies (tgds)

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

  $\varphi(\mathbf{x})$, $\psi(\mathbf{y})$ are conjunctions of atomic formulas.

- Extensive study of the boundary between decidability and undecidability of the implication problem for classes of egds and tgds in the 1970s and the 1980s.
  - For an overview, see

    "The Theory of Database Dependencies – A Survey"

    by R. Fagin and M.Y. Vardi – 1986.

Uses of Database Dependencies

- Over the years, equality-generating dependencies and tuple-generating dependencies have found many uses and applications in different areas of database research.
  - We will discuss such uses in data exchange and data integration.
- Moreover, equality-generating dependencies and tuple-generating dependencies are also encountered in unexpected places.

From Quantum Mechanics to Database Dependencies

Fast forward to 2013:

- *"Relational Hidden Variables and Non-Locality"*
  by S. Abramsky
- Study of the foundations of quantum mechanics in a relational framework.

Fact: Most properties formalized and studied by Abramsky can be expressed as either equality-generating dependencies or as tuple-generating dependencies.

From Quantum Mechanics to Database Dependencies

- Equality-generating dependencies
  - Weak Determinism (in fact, a key constraint)
  - Strong Determinism.

- Tuple-generating dependencies
  - No-signalling
  - λ-independence
  - Outcome independence
  - Parameter Independence
  - Locality

- **Example:** No-signalling for 2-dimensional relational models
  $\forall$ x,y,z,s,t,u,v (R(x,y,s,t) $\wedge$ R(x,z,u,v) $\rightarrow$ $\exists$ w R(x,z,s,w))

  "Whether an outcome s is possible for a given measurement x is independent of the other measurements."

Tutorial Outline

- Part II:
  - ✓ Functional Dependencies and Inclusion Dependencies
  - ✓ The Implication Problem for Database Dependencies

  - ❑ Data Inter-operability via Database Dependencies
  - ❑ Schema Mappings and Data Exchange
  - ❑ Managing Schema Mappings

A Different Use of Logic in Databases

In the past decade, logic has also been used is also used as a formalism to specify and study critical data interoperability tasks, such as

- Data integration

and

- Data exchange.

Tuple-generating dependencies have played a crucial role in this endeavor.

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:
  - Over $2B per year; 17% annual rate of growth.
  - Information integration consumes 40% of the budget of enterprise information technology shops.

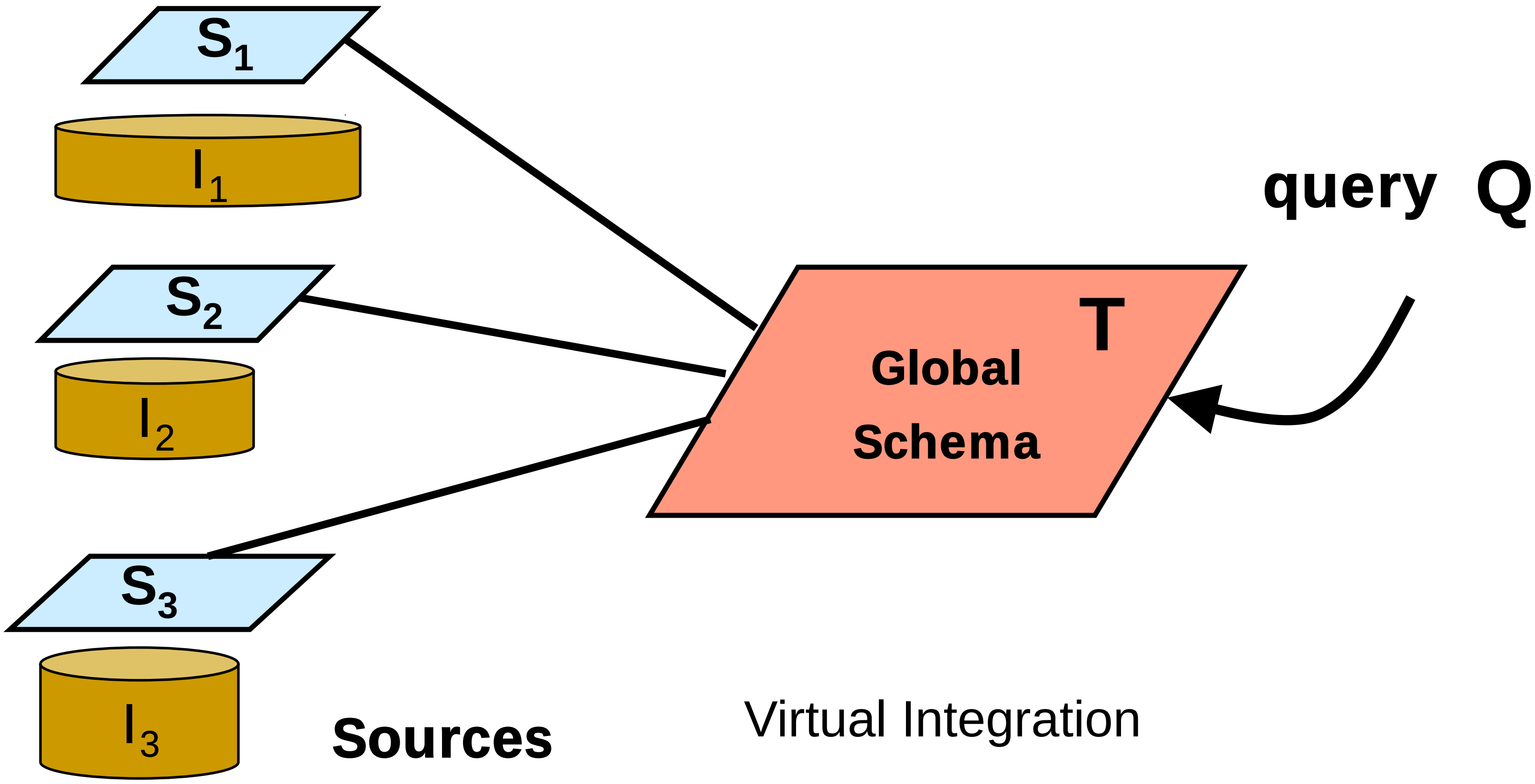
Two Facets of Information Integration

The research community has studied two different, but closely related, facets of information integration that have to do with data interoperability:

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

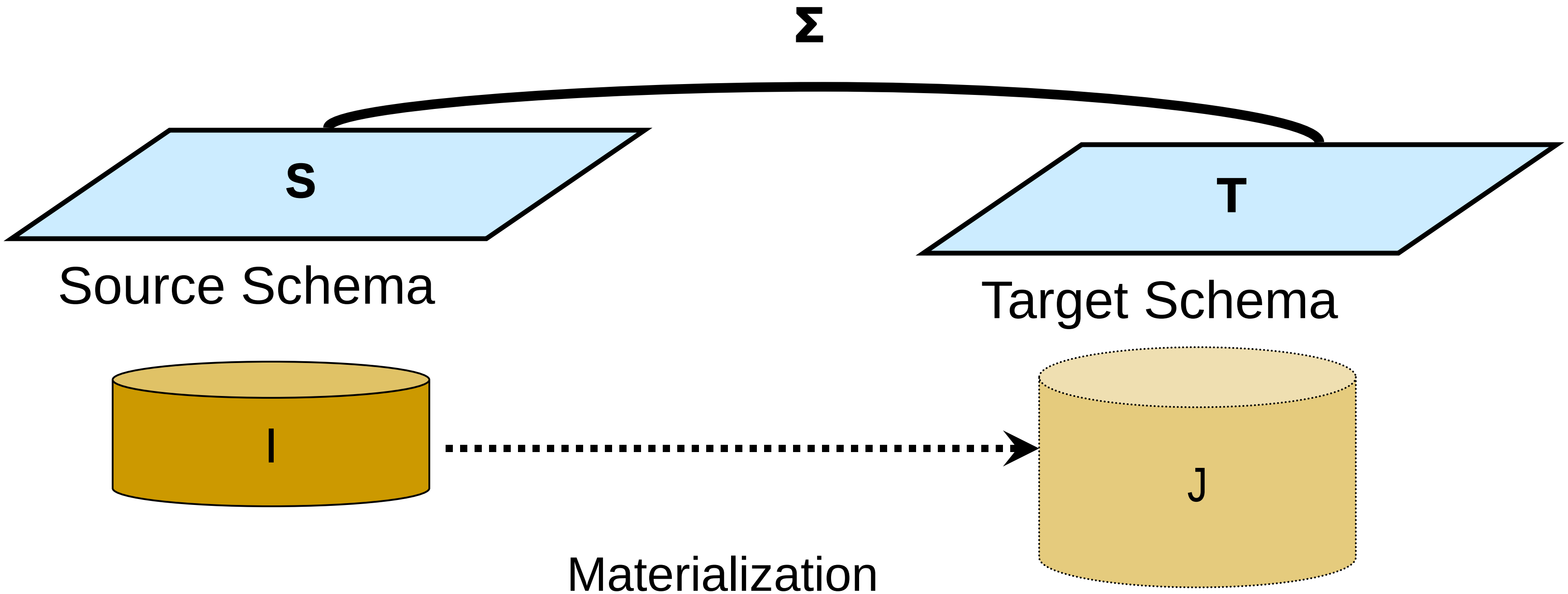
Data Integration

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.



Challenges in Data Interoperability

Fact:

- Data interoperability tasks require expertise, effort, and time.
- **Key challenge:** Specify the relationship between schemas.

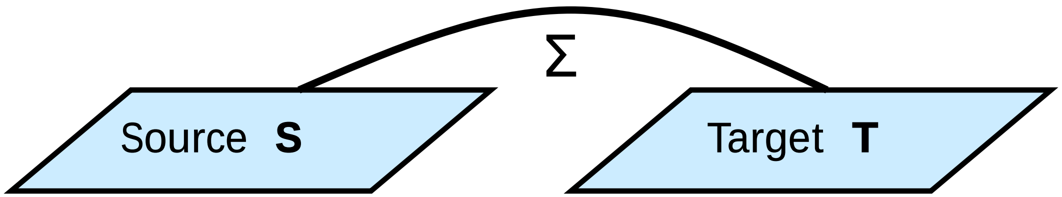
Earlier approach:

- Experts generate complex transformations that specify the relationship as programs or as SQL/XSLT scripts.
- Costly process, little automation.

More recent approach: Use **Schema Mappings**

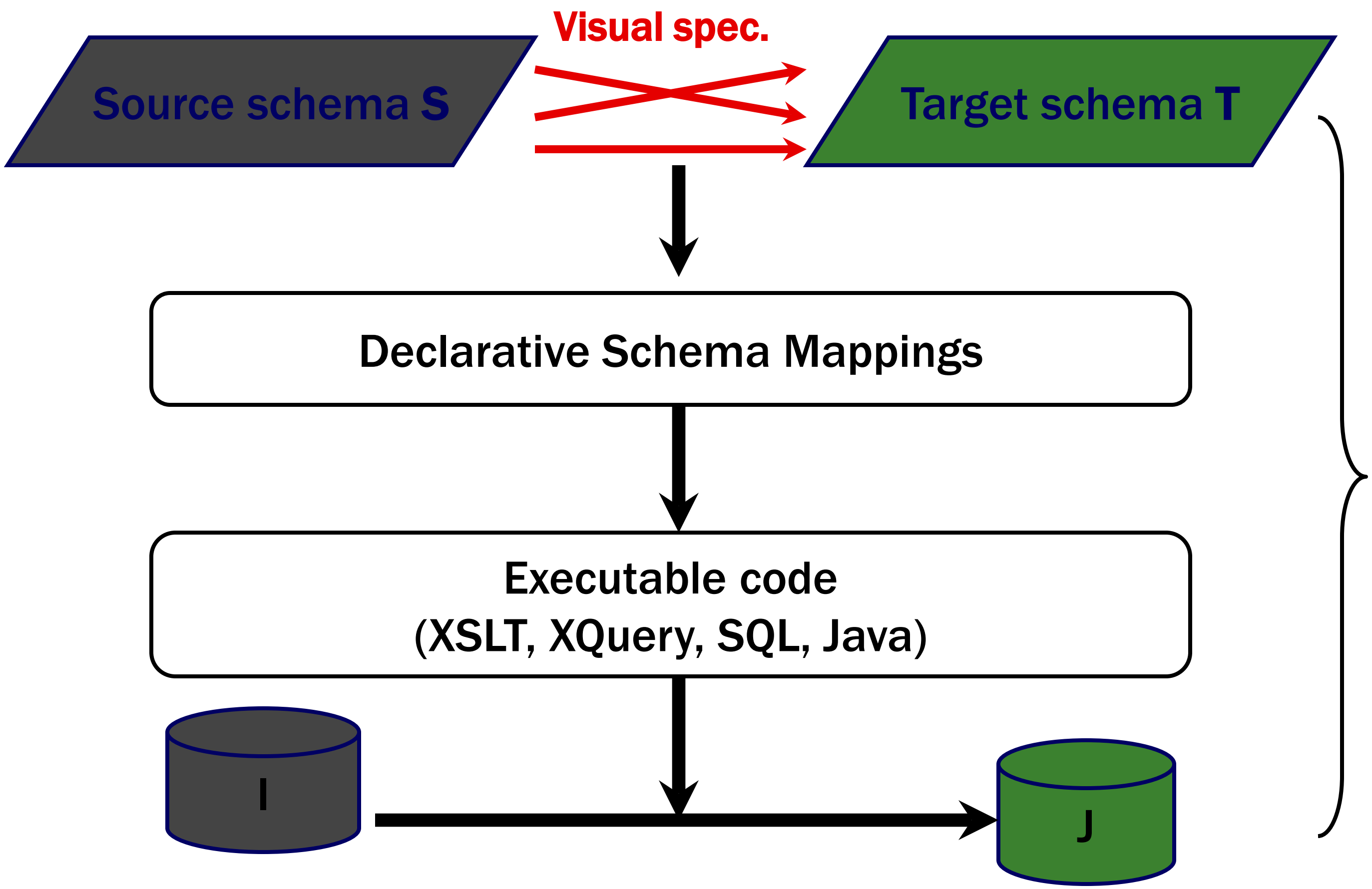
- Higher level of abstraction that separates the design of the relationship between schemas from its implementation.
- Schema mappings can be compiled into SQL/XSLT scripts automatically.

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**.
    - Typically, Σ is a finite set of formulas in some suitable logical formalism (*more on this later*).
- Schema mappings are the essential building blocks in formalizing data integration and data exchange.

Schema-Mapping Systems: State-of-the-Art



Generic architecture of schema-mapping systems
e.g.,
IBM Clio, HePToX
Altova MapForce
Stylus Studio
MS Biztalk Mapper

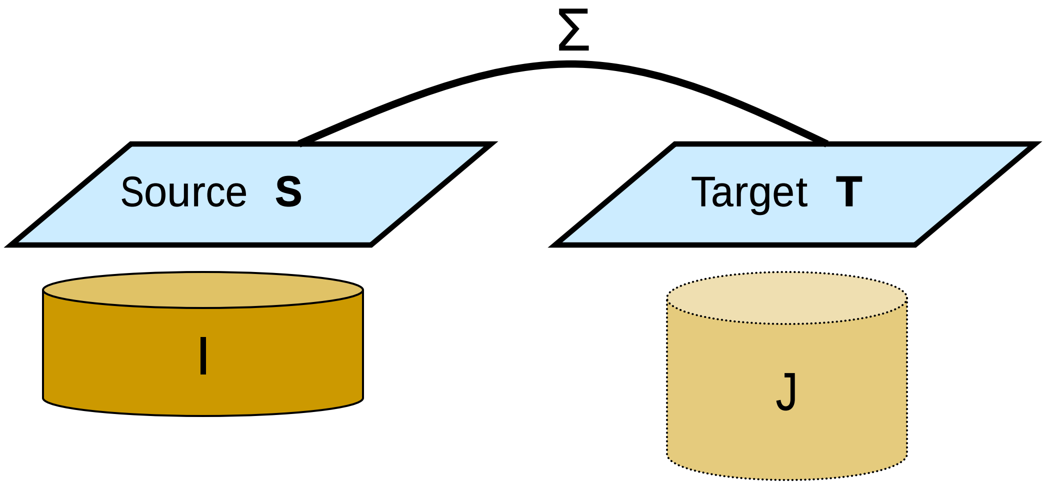
Acknowledgments

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

  Papers in ICDT, PODS, TCS, ACM TODS, JACM.

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

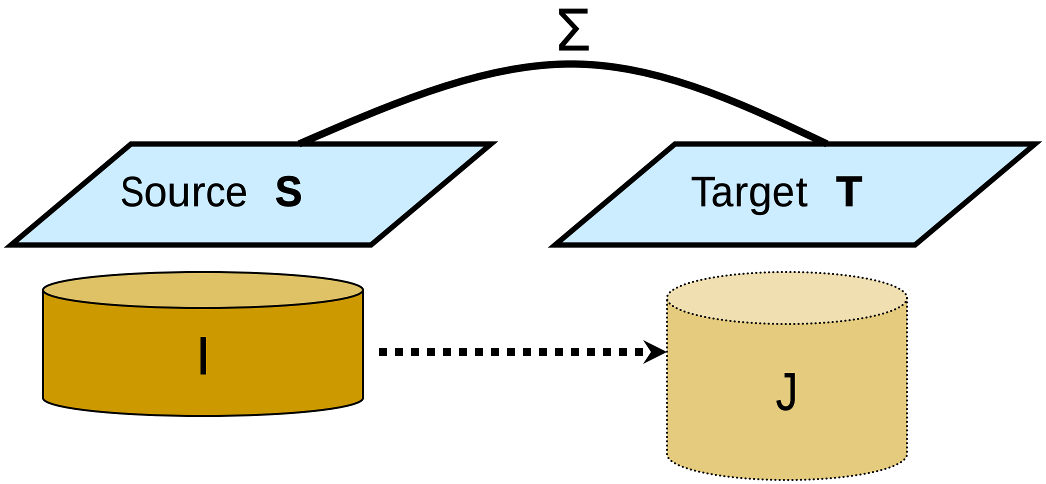
Schema Mappings



- 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) ² 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**.

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, …

Solutions in Schema Mappings

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

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

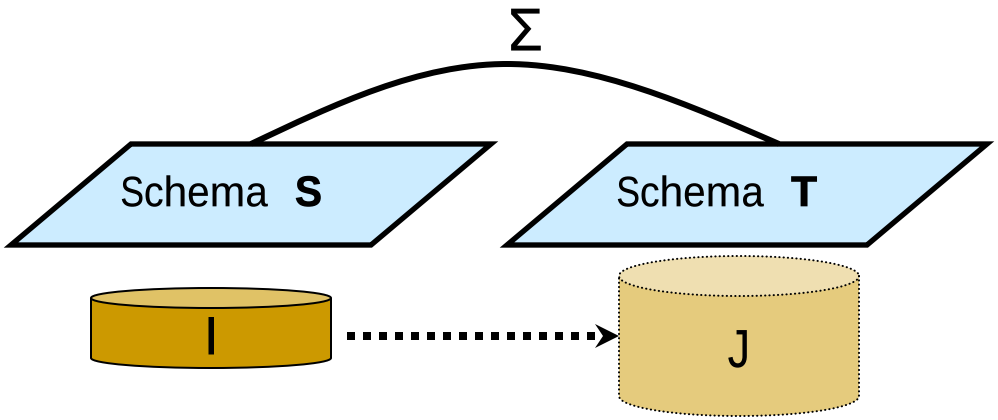
Fact: In general, for a given source instance I,

- 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.

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)

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

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)

  - Special Case: 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.

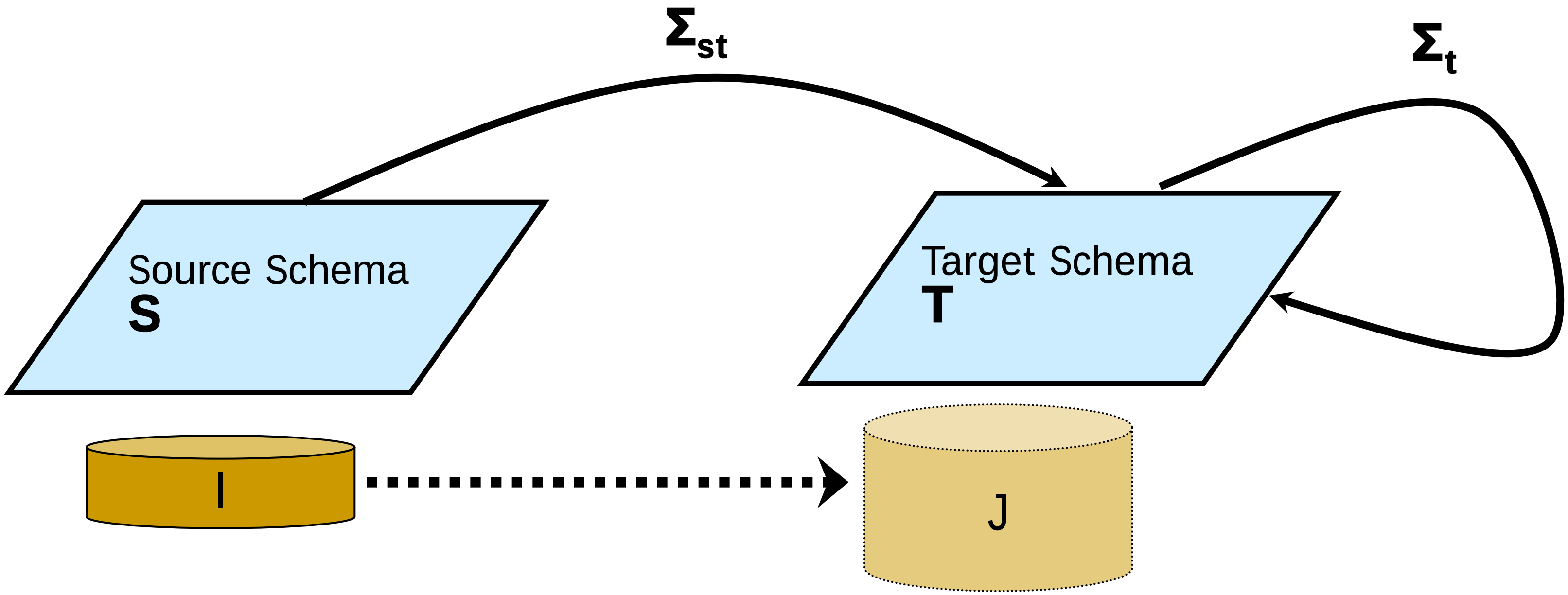
- Target Equality Generating Dependencies (egds):

  $$\varphi_T(\mathbf{x}) \rightarrow (x_1=x_2)$$

  $(\text{Mgr}(e, d_1) \wedge \text{Mgr}(e, d_2)) \rightarrow (d_1 = d_2)$

  (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?

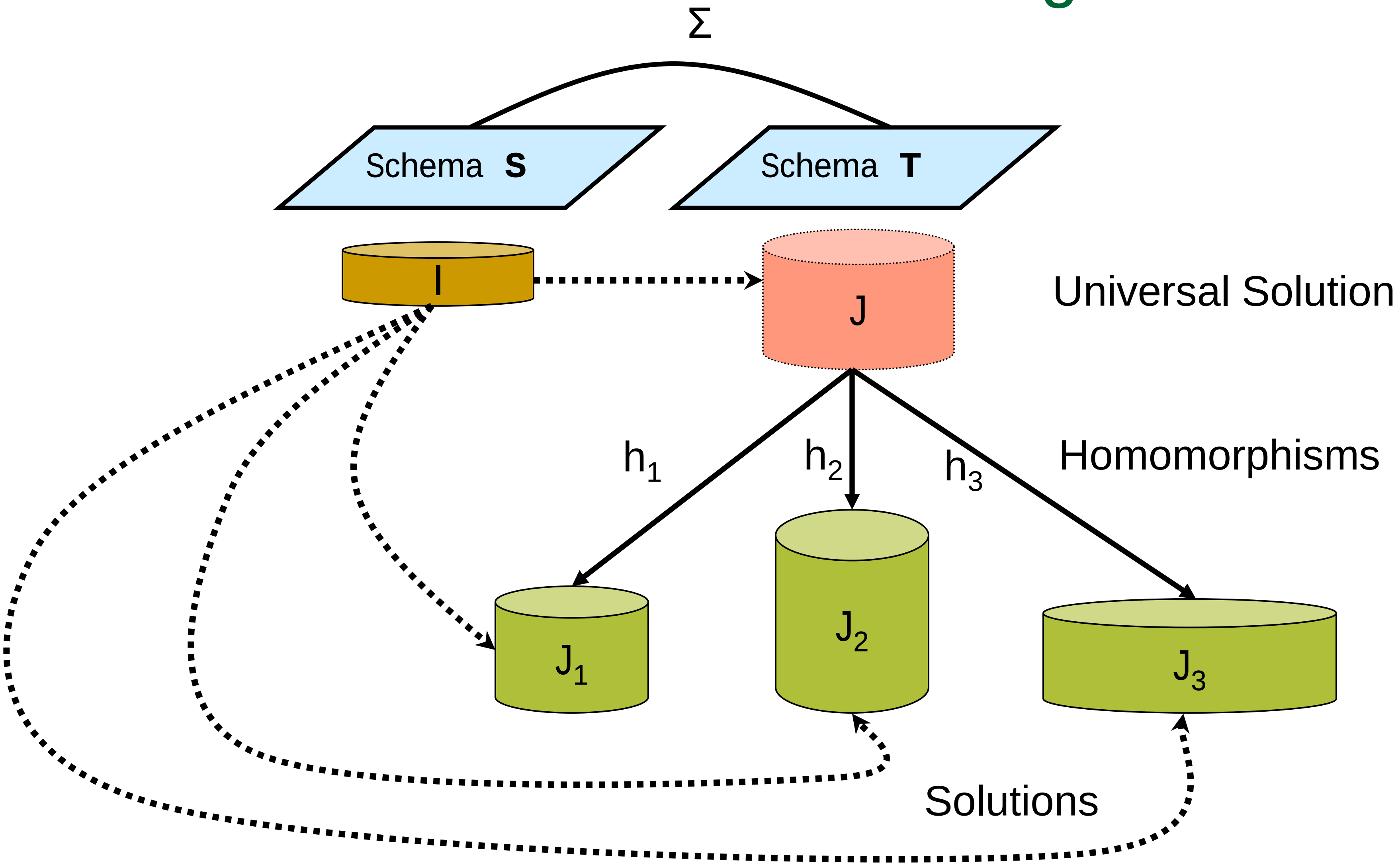
Universal Solutions in Data Exchange

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)

Σ_{st}: E(x,y) $\rightarrow$ $\exists$ z(F(x,z)Æ F(z,y))

Σ_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
    $R'(x,y,u) \wedge R'(y,z,v)\ \text{Æ}\ R'(u,z,w) \rightarrow R'(x,u,w)$

  - 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.

$$\begin{tikzcd} D.1 & M.1 \arrow[l] \arrow[r, shift right] & D.2 \arrow[l, shift right] \end{tikzcd}$$

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

$$\begin{tikzcd} E.1 & E.2 \arrow[l] \arrow[loop right] \end{tikzcd}$$

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

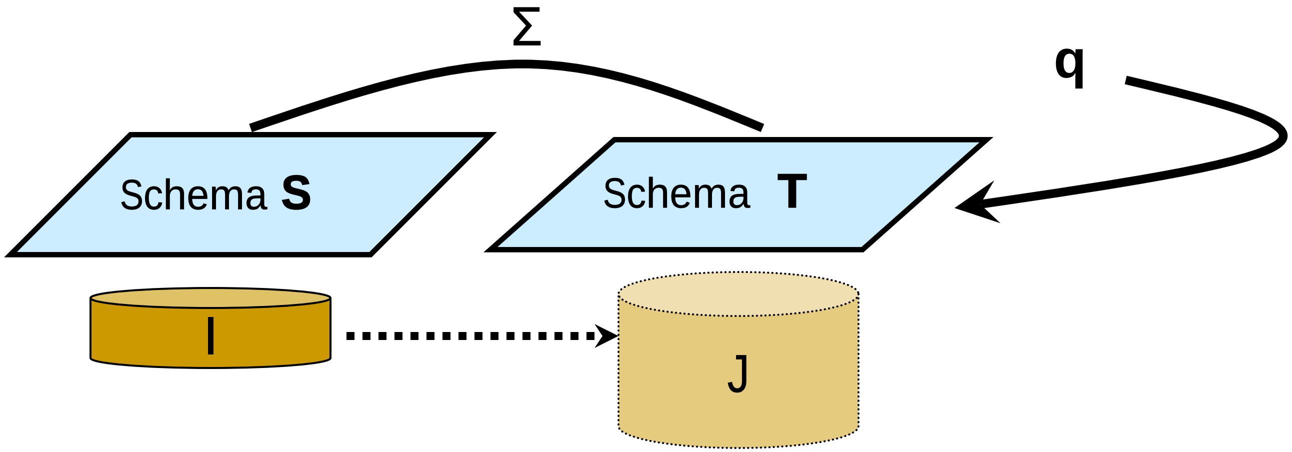
Summary: The existence-of-solutions problem

- 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).

Query Answering in Data Exchange



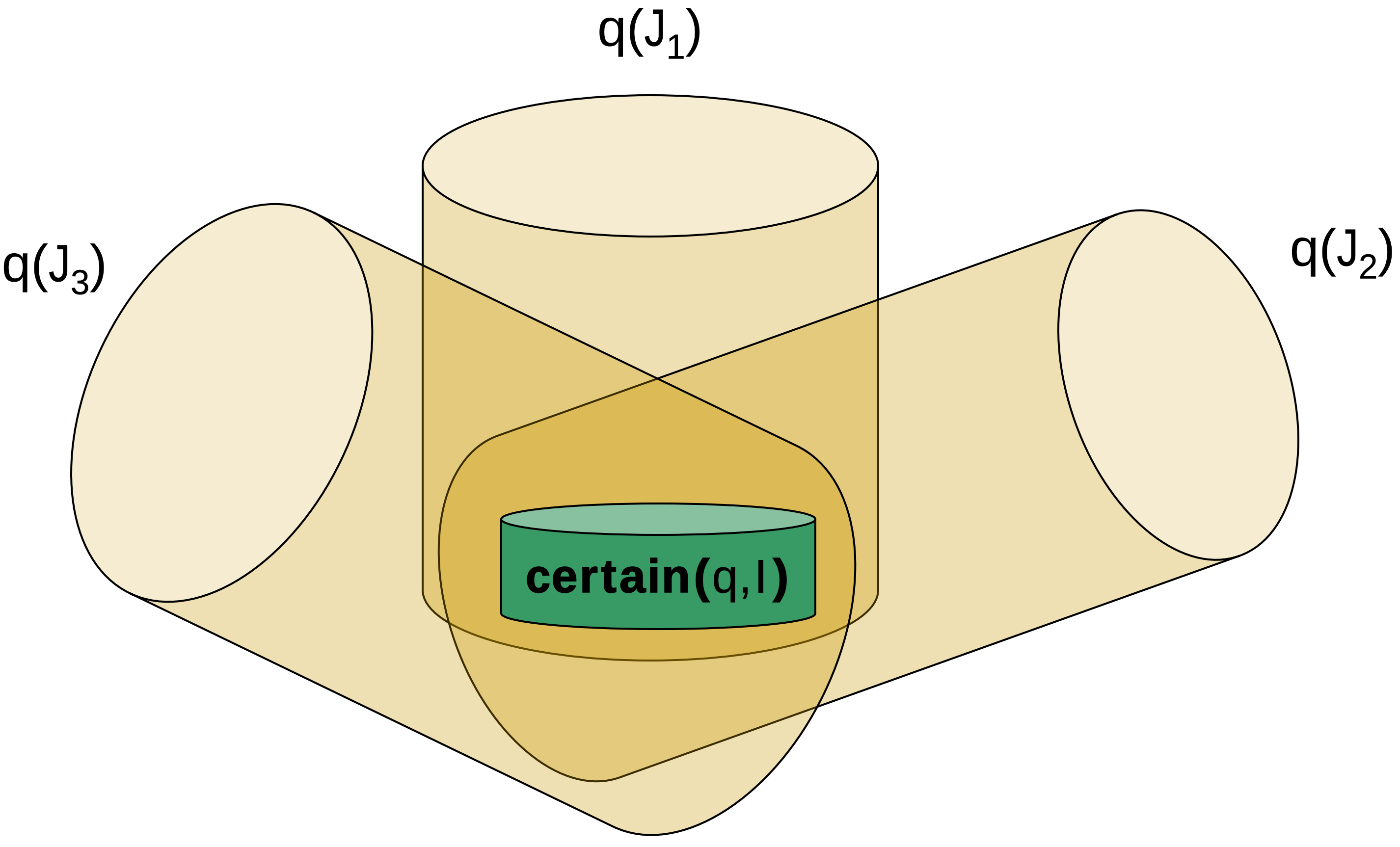
Question: What is the semantics of target query answering?

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**.

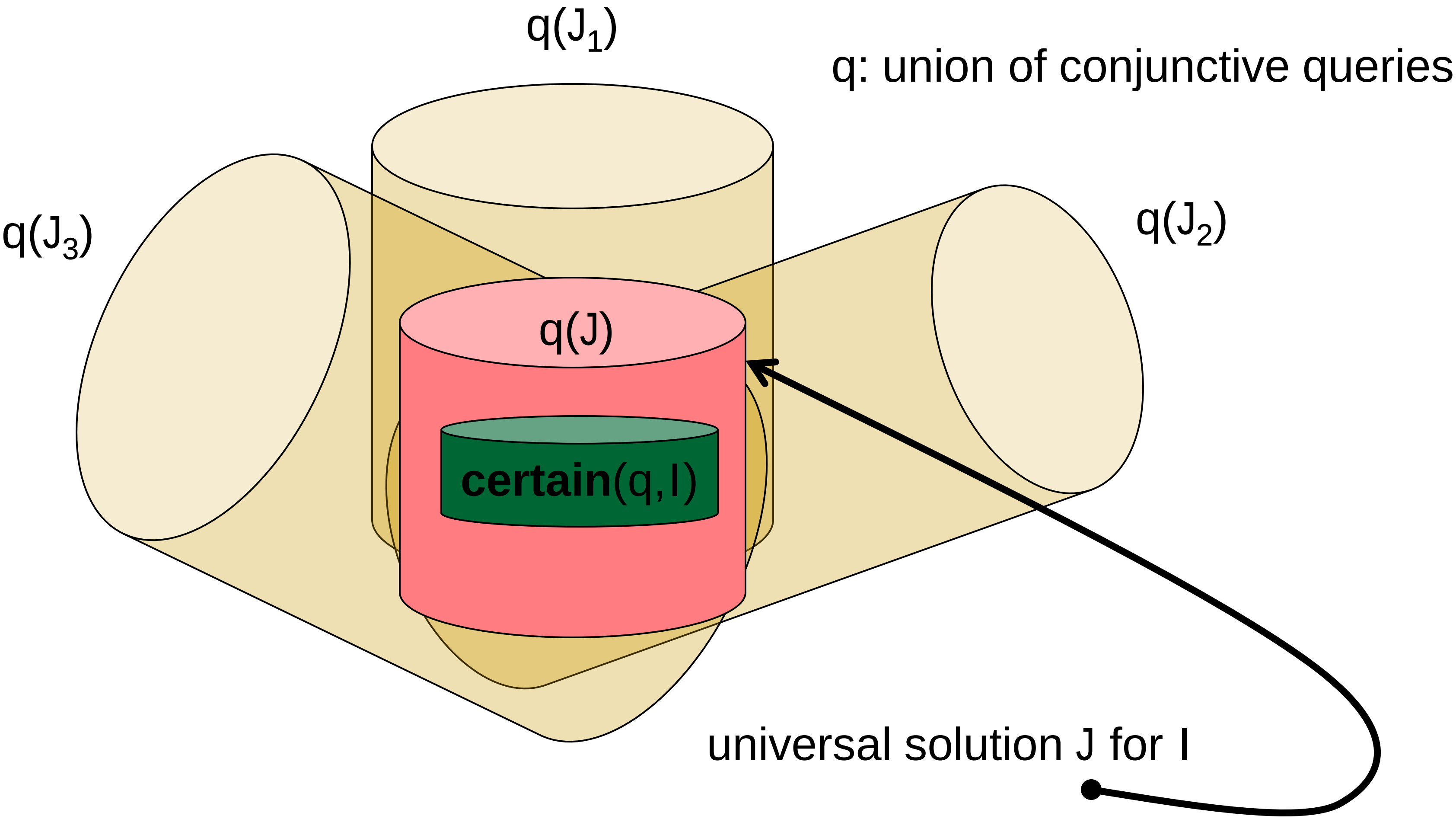
- If I is a source instance and J is a universal solution for I, then

  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 (**≠**).

- 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.

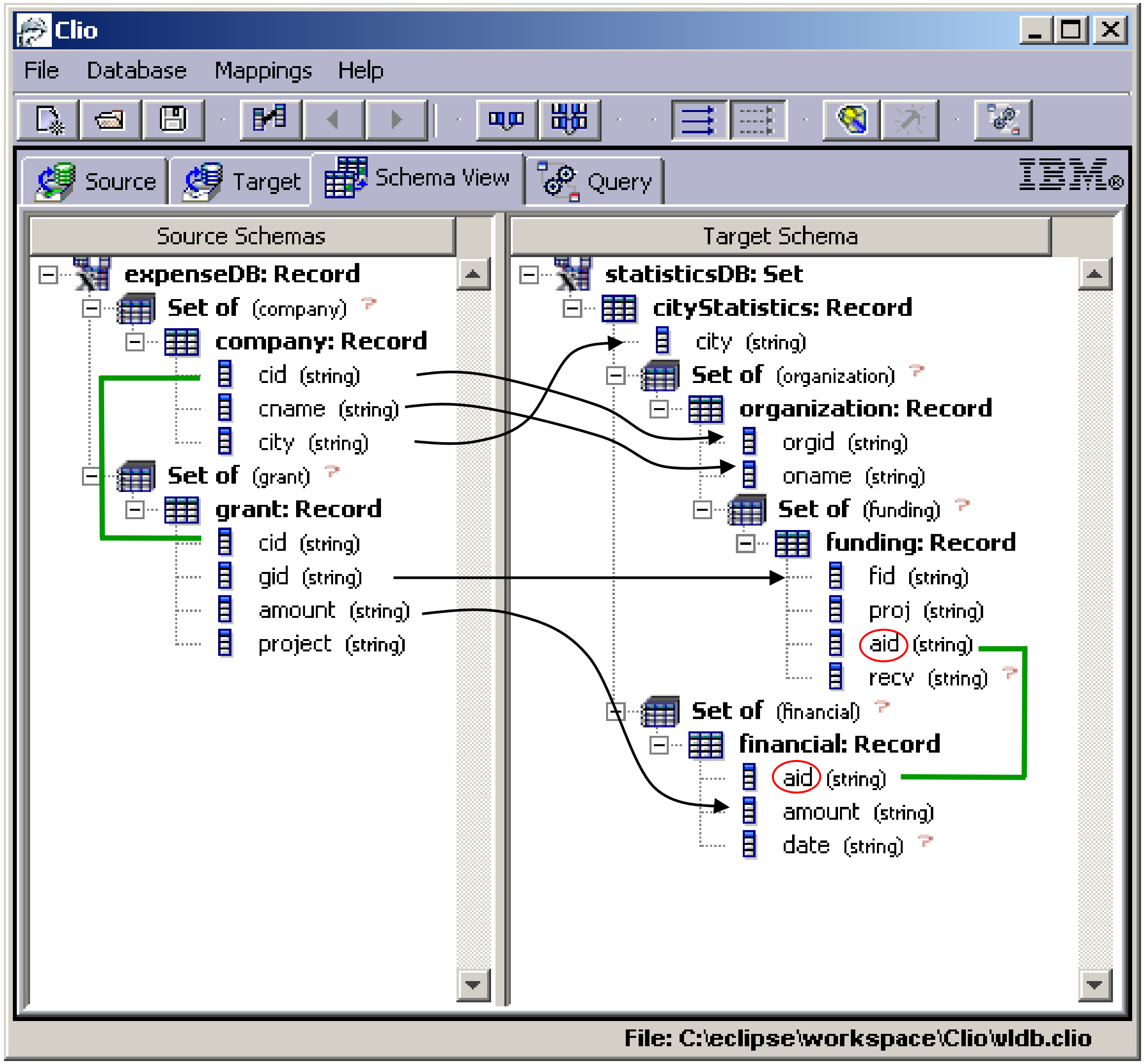
- Universal solutions used as the semantics of data exchange.

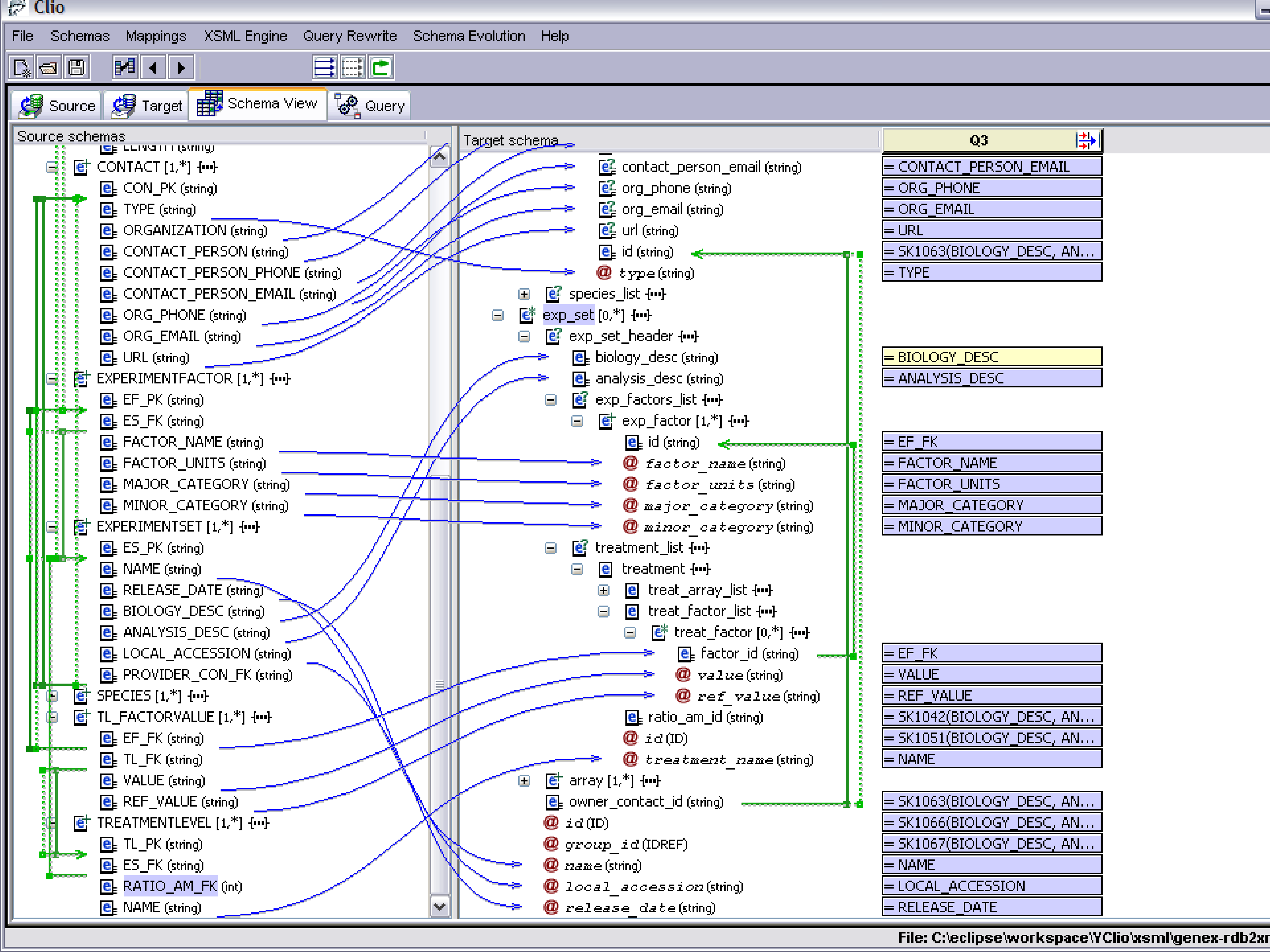
- Universal solutions are generated via SQL queries extended with Skolem functions (implementation of chase procedure), provided there are no target constraints.

- Clio technology is now part of
  IBM InfoSphere® Data Architect.

Some Features of Clio

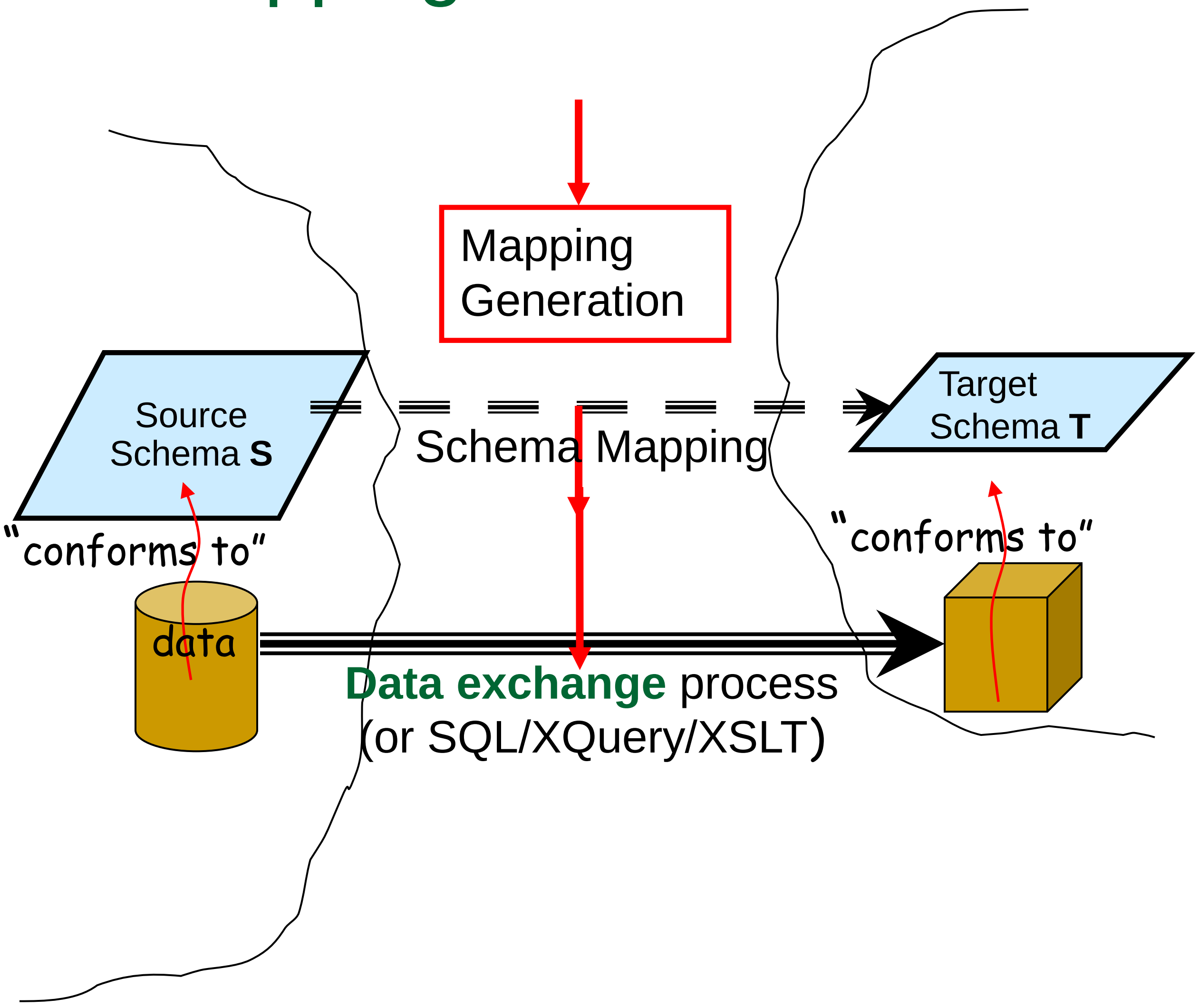
- Supports nested structures
  - Nested Relational Model
  - Nested Constraints
- Automatic & semi-automatic discovery of attribute correspondence.
- Interactive derivation of schema mappings.
- Performs data exchange





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



Schema Mappings (one of many pages)

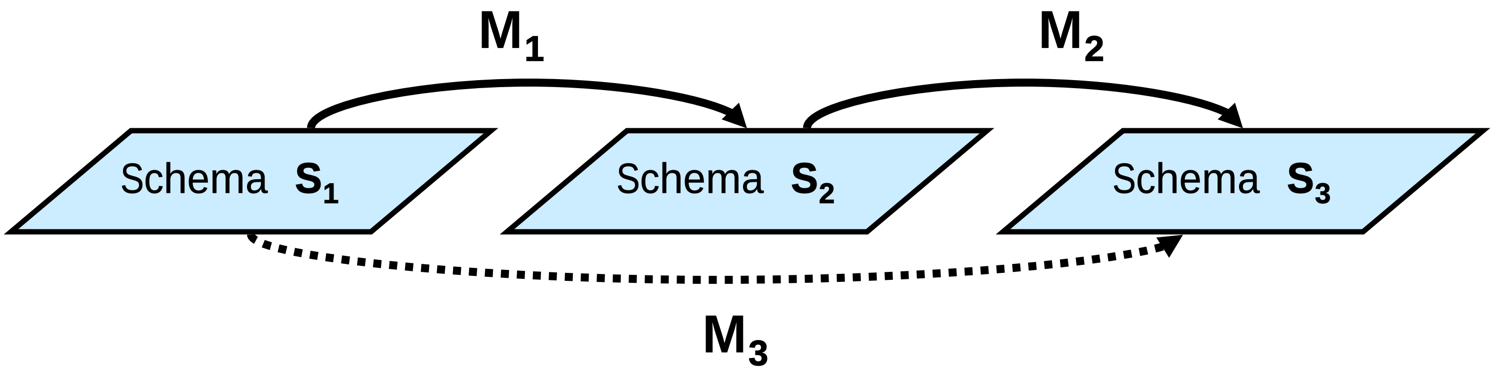
```
Map 2:
        for   sm2x0 in S0.dummy_COUNTRY_4
        exists  tm2x0 in S27.dummy_country_10, tm2x1 in S27.dummy_organiza_13
           where  tm2x0.country.membership=tm2x1.organization.id,
        satisf  sm2x0.COUNTRY.AREA=tm2x0.country.area, sm2x0.COUNTRY.CAPITAL=tm2x0.country.capital,
                sm2x0.COUNTRY.CODE=tm2x0.country.id, sm2x0.COUNTRY.NAME=tm2x0.country.name,
                sm2x0.COUNTRY.POPULATION=tm2x0.country.population,(
        Map 3:
                for  sm3x0 in S0.dummy_GEO_RIVE_23, sm3x1 in S0.dummy_RIVER_24,
                        sm3x2 in S0.dummy_PROVINCE_5
                   where  sm3x0.GEO_RIVER.RIVER=sm3x1.RIVER.NAME, sm3x2.PROVINCE.NAME=sm3x0.GEO_RIVER.PROVINCE,
                        sm3x2.PROVINCE.COUNTRY=sm2x0.COUNTRY.CODE,
                exists  tm3x0 in S27.dummy_river_24, tm3x1 in tm3x0.river.dummy_located_23,
                        tm3x4 in S27.dummy_country_10, tm3x5 in tm3x4.country.dummy_province_9,
                        tm3x6 in S27.dummy_organiza_13
                where  tm3x4.country.membership=tm3x6.organization.id, tm3x5.province.id=tm3x1.located.province,
                        tm2x0.country.id=tm3x1.located.country,
                satisf  sm2x0.COUNTRY.AREA=tm3x4.country.area, sm2x0.COUNTRY.CAPITAL=tm3x4.country.capital,
                        sm2x0.COUNTRY.CODE=tm3x4.country.id, sm2x0.COUNTRY.NAME=tm3x4.country.name,
                        sm2x0.COUNTRY.POPULATION=tm3x4.country.population, sm3x1.RIVER.LENGTH=tm3x0.river.length,
                        sm3x0.GEO_RIVER.COUNTRY=tm3x1.located.country, sm3x0.GEO_RIVER.PROVINCE=tm3x1.located.province,
                        sm3x1.RIVER.NAME=tm3x0.river.name ),(
        Map 4:
                for  sm4x0 in S0.dummy_GEO_ISLA_25, sm4x1 in S0.dummy_ISLAND_26,
                        sm4x2 in S0.dummy_PROVINCE_5
                   where  sm4x0.GEO_ISLAND.ISLAND=sm4x1.ISLAND.NAME, sm4x2.PROVINCE.NAME=sm4x0.GEO_ISLAND.PROVINCE,
                        sm4x2.PROVINCE.COUNTRY=sm2x0.COUNTRY.CODE,
                exists  tm4x0 in S27.dummy_island_26, tm4x1 in tm4x0.island.dummy_located_25,
                        tm4x4 in S27.dummy_country_10, tm4x5 in tm4x4.country.dummy_province_9,
                        tm4x6 in S27.dummy_organiza_13
                   where  tm4x4.country.membership=tm4x6.organization.id, tm4x5.province.id=tm4x1.located.province,
                        tm2x0.country.id=tm4x1.located.country,
                satisf  sm2x0.COUNTRY.AREA=tm4x4.country.area, sm2x0.COUNTRY.CAPITAL=tm4x4.country.capital,
                        sm2x0.COUNTRY.CODE=tm4x4.country.id, sm2x0.COUNTRY.NAME=tm4x4.country.name,
                        sm2x0.COUNTRY.POPULATION=tm4x4.country.population, sm4x1.ISLAND.AREA=tm4x0.island.area,
                        sm4x1.ISLAND.COORDINATESLAT=tm4x0.island.latitude, sm4x0.GEO_ISLAND.COUNTRY=tm4x1.located.country,
                        sm4x0.GEO_ISLAND.PROVINCE=tm4x1.located.province, sm4x1.ISLAND.COORDINATESLONG=tm4x0.island.longitude,
                        sm4x1.ISLAND.NAME=tm4x0.island.name ),(
        Map 5:
                for  sm5x0 in S0.dummy_GEO_SEA_19, sm5x1 in S0.dummy_SEA_20,
                        sm5x2 in S0.dummy_PROVINCE_5
                   where  sm5x2.PROVINCE.NAME=sm5x0.GEO_SEA.PROVINCE, sm5x0.GEO_SEA.SEA=sm5x1.SEA.NAME,
                                sm5x2.PROVINCE.COUNTRY=sm2x0.COUNTRY.CODE,
                exists  tm5x0 in S27.dummy_sea_19, tm5x1 in tm5x0.sea.dummy_located_18,
                        tm5x4 in S27.dummy_country_10, tm5x5 in tm5x4.country.dummy_province_9,
                        tm5x6 in S27.dummy_organiza_13
                   where  tm5x4.country.membership=tm5x6.organization.id, tm5x5.province.id=tm5x1.located.province,
                        tm2x0.country.id=tm5x1.located.country,
                satisf  sm2x0.COUNTRY.AREA=tm5x4.country.area, sm2x0.COUNTRY.CAPITAL=tm5x4.country.capital,
                        sm2x0.COUNTRY.CODE=tm5x4.country.id, sm2x0.COUNTRY.NAME=tm5x4.country.name,
                        sm2x0.COUNTRY.POPULATION=tm5x4.country.population, sm5x1.SEA.DEPTH=tm5x0.sea.depth,
                        sm5x0.GEO_SEA.COUNTRY=tm5x1.located.country, sm5x0.GEO_SEA.PROVINCE=tm5x1.located.province,
                        sm5x1.SEA.NAME=tm5x0.sea.name ),(
```

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

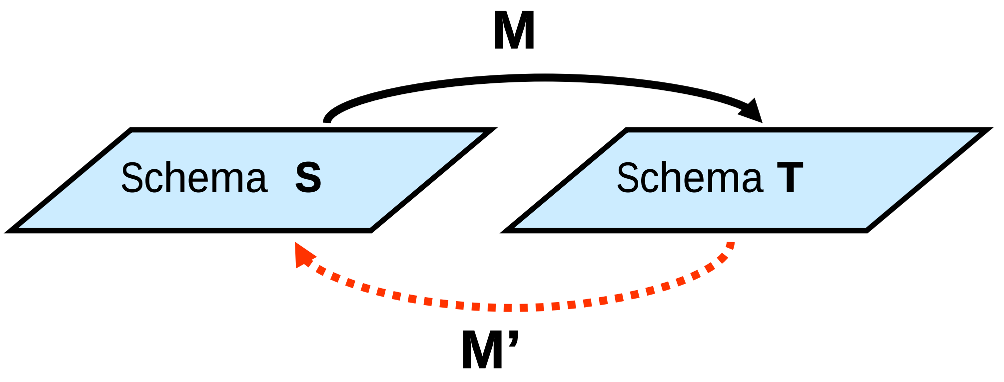
Composing Schema Mappings



- 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

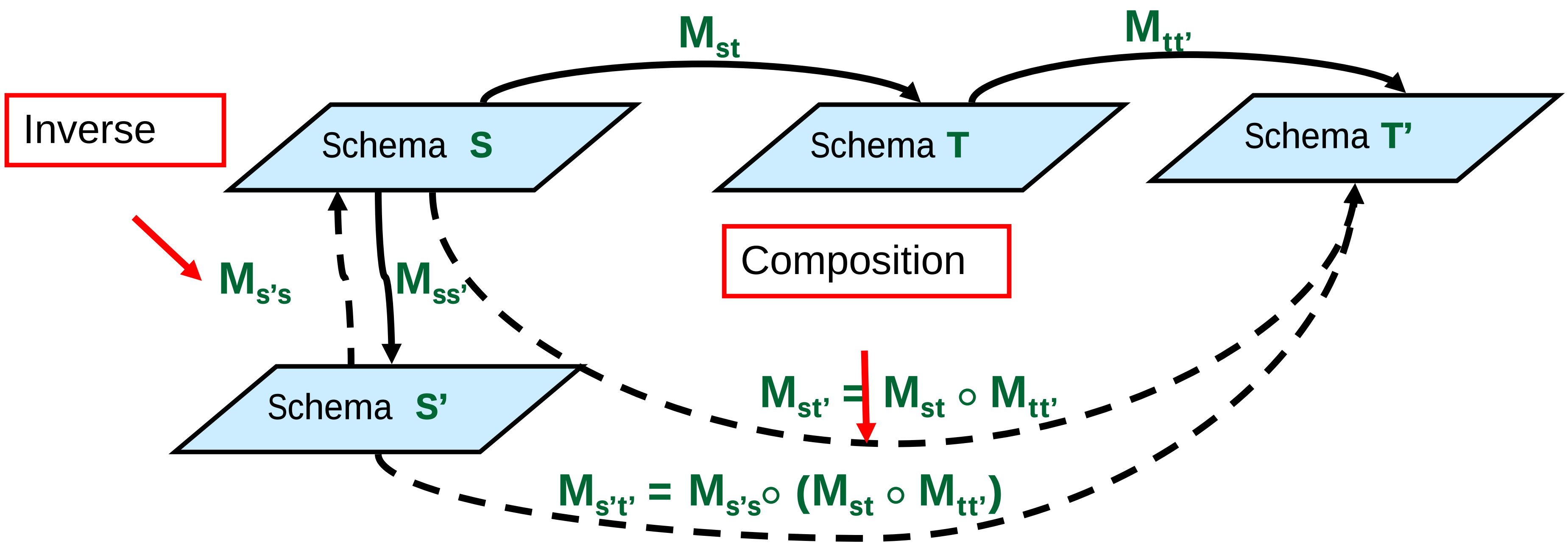


- Given **M**, derive **M’** that “undoes” **M**

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?

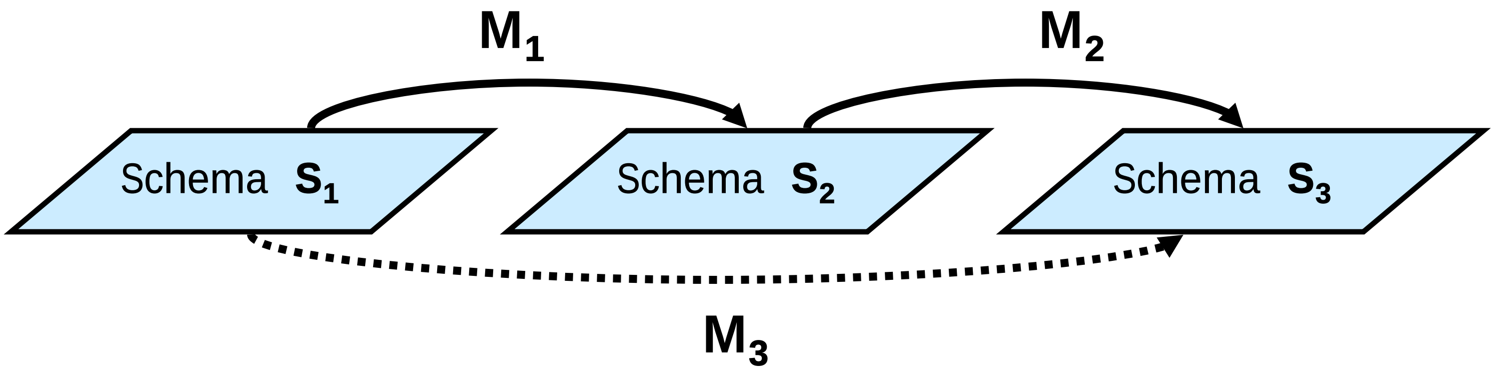
- **Language:**

  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

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

- **Definition:**

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

$\mathrm{Inst}(\mathbf{M_3}) = \mathrm{Inst}(\mathbf{M_1}) \circ \mathrm{Inst}(\mathbf{M_2})$, 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

Issues in Composition of Schema Mappings

- The semantics of composition was the first main issue.
- The language of composition is the second main issue:
  - Is the language of s-t tgds *closed under composition*?

  If $\mathbf{M}_{12}$ and $\mathbf{M}_{23}$ are specified by finite sets of s-t tgds, is $\mathbf{M}_{12} \circ \mathbf{M}_{23}$ also specified by a finite set of s-t tgds?

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

Terminogical Reminder: GLAV, GAV, and LAV

- s-t tgds or GLAV (global-and-local-as-view) constraints:

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

  where $\varphi(\mathbf{x})$ and $\psi(\mathbf{x})$ are conjunctions of atoms.
- GAV (global-as-view) constraint:

  $\forall \mathbf{x} \, (\varphi(\mathbf{x}) \rightarrow R(\mathbf{x})),$ where R is a target relation.
- 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.
  - Strict LAV (strict 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 and each variable occurs ***only once*** in $P(\mathbf{x})$.

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.
- If $\mathbf{M_1}$ is a GLAV schema mapping and $\mathbf{M_2}$ is a strict LAV 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
- GLAV $\circ$ strict LAV = 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 $\not\subseteq$ LFP

- **M_1** : LAV schema mapping
  $\forall x\, \forall y\, (E(x,y) \rightarrow \exists u \exists v\, (C(x,u) \wedge C(y,v)))$
  $\forall x\, \forall y\, (E(x,y) \rightarrow F(x,y))$
- **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 $(I_1, I_3) \in \text{Inst}(\mathbf{M_1}) \circ \text{Inst}(\mathbf{M_2})$

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

The Language of Composition

Question:

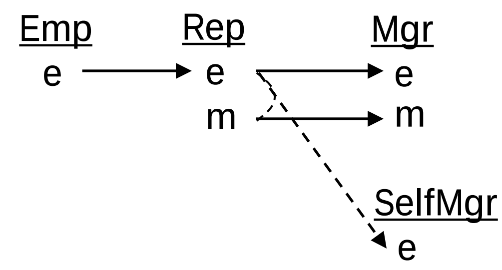
What is the “right” language for expressing the composition of two GLAV schema mappings?

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 and LAV schema mapping
- (but **not** strictly LAV)
  - $\forall e\ \forall m\ (Rep(e,m) \rightarrow Mgr(e,m))$
  - $\forall e\ (Rep(e,e) \rightarrow SelfMgr(e))$



- **Theorem: $M_1 \circ M_2$** 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

- **M_1** : LAV schema mapping
  - $\forall e\ (Emp(e) \rightarrow \exists m\ Rep(e,m))$
- **M_2** : GAV and LAV (but **not** strict LAV) schema mapping
  - $\forall e\ \forall m\ (Rep(e,m) \rightarrow Mgr(e,m))$
  - $\forall e\ (Rep(e,e) \rightarrow SelfMgr(e))$

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

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

$\forall e\ (\mathrm{Emp}(e) \wedge (e = \mathbf{f}(e)) \rightarrow \mathrm{SelfMgr}(e))).$

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

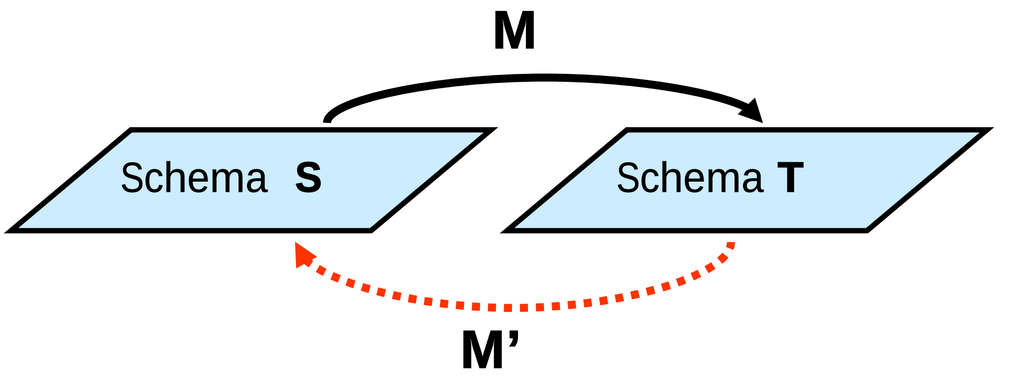
- GAV $\circ$ GAV = GAV
- GAV $\circ$ GLAV = GLAV
- GLAV $\circ$ strict LAV = GLAV.
- GLAV $\circ$ GLAV $\not\subseteq$ GLAV. In fact, even
  - LAV $\circ$ GAV $\not\subseteq$ GLAV.
  - LAV $\circ$ strict LAV $\not\subseteq$ GLAV.
- GLAV $\circ$ GLAV = SO-tgds = SO-tgds $\circ$ SO-tgds

- SO-tgds are the "right" language for composing GLAV schema mappings.
- SO-tgds are "chasable": Universal solutions computable in PTIME.
- SO-tgds and the composition algorithm have been incorporated in the IBM InfoSphere® Data Architect.

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
- Local transformations and conjunctive-query equivalence
  Fagin and K ...   - 2012

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
  - Chase inverse of schema mappings
    Fagin, K ..., Popa, Tan – 2011

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

Some Directions of Research

- Settle the semantics of the inverse operator and identify the "right" language for the inverse operator.
- Schema mappings specified by more expressive languages:
  - GLAV constraints with arithmetic operations
    - Joint work with ten Cate and Othman – in EDBT 2013
    - Connections with the existential theory of the reals
  - Disjunctive GLAV constraints
  - ...
- Deriving schema mappings from data examples
  - Currently under active investigation (with Alexe, ten Cate, Tan)
  - Connections with duality in constraint satisfaction.
- Applications of schema-mapping operators to:
  - Analysis of schema evolution.