Schema Mappings and Data Exchange

Lecture #5

EASSLC 2012

Southwest University

August 2012

Logic and Databases

Two uses of logic in databases:

- Logic as a query language.
- Logic as a specification language for expressing integrity constraints (semantic restrictions) that the databases of interest must obey. Integrity constraints are also known as database dependencies.

Functional Dependencies

Definition: Let R be a relational schema and r an instance of R.

- If $A_1,\ldots,A_m$, B are attributes of R, then we say that

  r satisfies the functional dependency

  $$A_1,\ldots,A_m \rightarrow B$$

  if whenever two tuples in r agree on the values of $A_1,\ldots,A_m$, then they also agree on the value of B.

  (in other words, there are no two tuples in r that have the same value on the attributes of $A_1,\ldots,A_m$, but differ on the value of B).
- If $A_1,\ldots,A_m$, $B_1,\ldots,B_k$ are attributes of R, then we say that

  r satisfies the functional dependency

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

  if r satisfies the functional dependencies

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

  ...

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

Functional Dependencies

- Definition: A relational schema R satisfies the functional dependency

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

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

- Fact: In effect, the above definition imposes a semantic restriction on the instances of R, namely, we disallow all instances that violate the functional dependency $A_1,\ldots,A_m \rightarrow B_1,..,B_k$.

- Example: ENROLLS(student, course, term, grade)
  - student, course, term $\rightarrow$ grade (should be true)
  - student, term $\rightarrow$ course (should not be true)

Functional Dependencies

Question: How do we know that a FD holds for a database schema?

Answer:

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

- A FD may be derived (inferred) from other known FDs about the schema.

Functional Dependencies

Example: COMPANY(employee, dpt, manager)

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

Functional Dependencies and Relational Calculus

Fact: Every functional dependency $A_1,\ldots,A_m \rightarrow B$ can be expressed in relational calculus. More formally, there is a relational calculus formula ψ such that for every database instance r, we have that the following are equivalent:

- $r \models A_1,\ldots,A_m \rightarrow B$
- $r \models \psi$.

Proof (by example): Assume that R has attributes A,B,C,D.
Then the following are equivalent for the FD $A,B \rightarrow C$.

- $r \models A,B \rightarrow C$.
- $r \models \forall x,y,z,w,z',w'(R(x,y,z,w) \wedge R(x,y,z',w') \rightarrow z = z')$.

Note: The formula $\forall x,y,z,w,z',w'(R(x,y,z,w) \wedge R(x,y,z',w') \rightarrow z = z')$ is an example of an equality-generating dependency (egd).

Equality-Generating Dependencies

Definition: An equality-generating dependency (egd) is a formula of relational calculus of the form:

$$\forall\, x_1,\ldots,x_n(\varphi(x_1,\ldots,x_n) \rightarrow x_i = x_j),$$

where $\varphi(x_1,\ldots,x_n)$ is a conjunction of atomic formulas
(i.e., φ is a conjunctive query)

Examples:

- $\forall x_1,x_2,x_3(R(x_1,x_2) \wedge P(x_2,x_3) \wedge T(x_2) \rightarrow x_2 = x_3)$
  - This is an egd, but not a FD.

- $\forall x_1,x_2,x_3(R(x_1,x_2) \wedge R(x_1,x_3) \rightarrow x_2 = x_3)$
  - This is both an egd and a FD, namely $A_1 \rightarrow A_2$.

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.

- A database instance r 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 of the shema satisfies this ID.

Inclusion Dependencies and Relational Calculus

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

Proof (by example): Consider the ID

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

which expresses the integrity constraint:

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

This ID is equivalent to the relational calculus formula

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

Note: The formula $\forall x,y,z$ (ENROLLS(x,y,z) $\rightarrow$ $\exists w$ PERFORM(x,z,w)) is an example of a tuple-generating dependency (tgd).

Tuple-Generating Dependencies

Definition: A tuple-generating dependency (tgd) is a formula of relational calculus of the form:

$$\forall x_1,\ldots,x_n(\varphi(x_1,\ldots,x_n) \rightarrow \exists y_1,..,y_m\ \psi(x'_1,\ldots,x'_k,y_1,\ldots,y_m)),$$

where

- $\varphi(x_1,\ldots,x_n)$ and $\psi(x'_1,\ldots,x'_k,y_1,\ldots,y_m$ are conjunctions of atomic formulas
- The variables $x'_1,\ldots,x'_k$ are among the variables $x_1,\ldots,x_n$.

Note: In effect, a tuple-generating dependency asserts that one conjunctive query (namely, the one defined by $\varphi(x_1,\ldots,x_n)$) is contained in another conjunctive query (namely, the one defined by $\exists y_1,..,y_m\psi(x'_1,\ldots,x'_k,y_1,\ldots,y_m)$).

Tuple-Generating Dependencies

Examples:

- Every inclusion dependency is a tuple-generating dependency.

- $\forall$ x,y,z (E(x,y)$\wedge$ E(y,z) $\rightarrow$ E(x,z))
  - This is a tgd, but not an ID. It asserts that E is transitive.

- $\forall$ x,y(E(x,y) $\rightarrow \exists$ z(F(x,z) $\wedge$ F(z,y)).
  - This says that for every edge in E, there is a path of length 2 in F.
- $\forall$ x,y,z (P(x,y,z) $\rightarrow$ R(x,y)$\wedge$ T(y,z))
  - This says that P is decomposed to R and T.

Embedded Implicational Dependencies

Definition: A database integrity constraint is an embedded implicational dependency if it is either a tuple-generating dependency or an equality-generating dependency.

Fact: Embedded implicational dependencies contain as special cases the various classes of integrity constraints studied in the 1970s and the early 1980s, such as:

- Functional dependencies
- Join dependencies
- Inclusion dependencies.
- Multivalued dependencies.

(see the survey paper on database dependencies by Fagin and Vardi)

Relational Calculus in Databases

Note:

- Relational calculus has been used in databases in two different ways:
  - As a database query language
  - As a constraint language for specifying integrity constraints, e.g.,
    - Key constraints can be expressed in relational calculus.
    - Inclusion dependencies can be expressed in relational calculus.

- In what follows, we will see that relational calculus is also used to formalize critical data interoperability tasks, such as
  - Data integration and
  - Data exchange

The Data Interoperability Challenge

- Data may reside
  - at several different sites
  - in several different formats (relational, XML, …).
- Applications need to access and process all these data.
- Growing market of enterprise data interoperability tools: in particuclar, IBM, SAP, Oracle, and Microsoft offer competing software systems for data interoperability tasks.

Theoretical Aspects of Data Interoperability

The research community has studied two different, but closely related, facets of data interoperability:

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

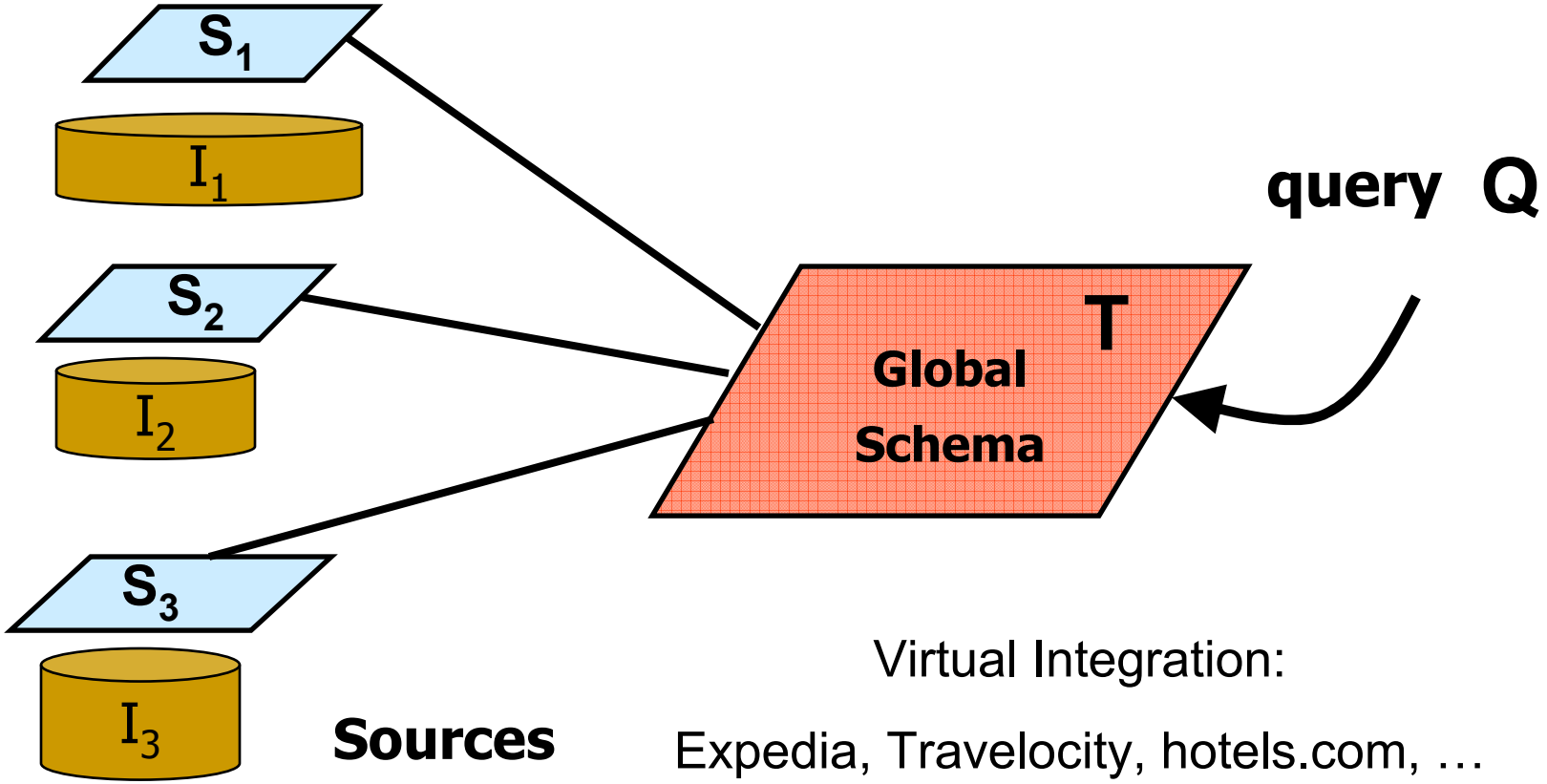
- **Data Exchange** (aka **Data Translation**)
  - Formalized and studied for the past 7-8 years

    "*Data exchange is the oldest database problem*"

    Phil Bernstein - 2003

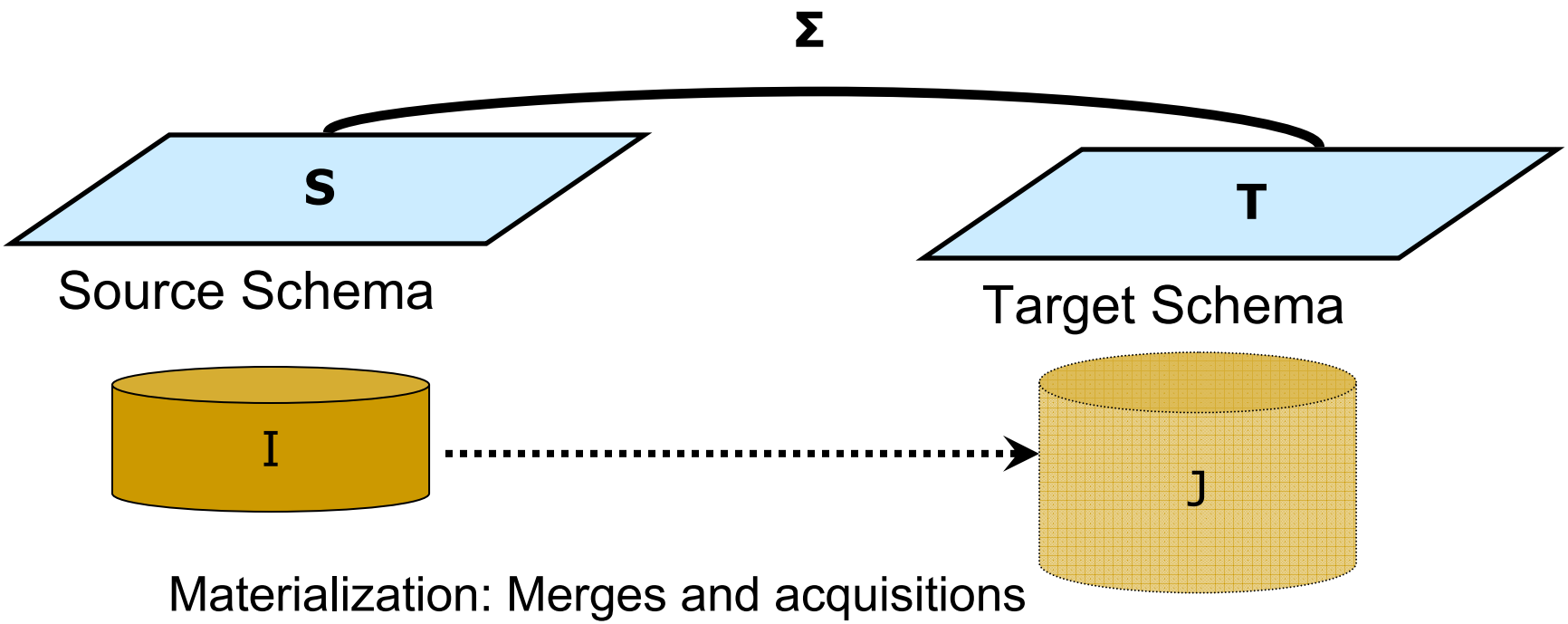
Data Integration

Query heterogeneous data in different sources via a virtual global schema



Data Exchange

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



Schema Mappings

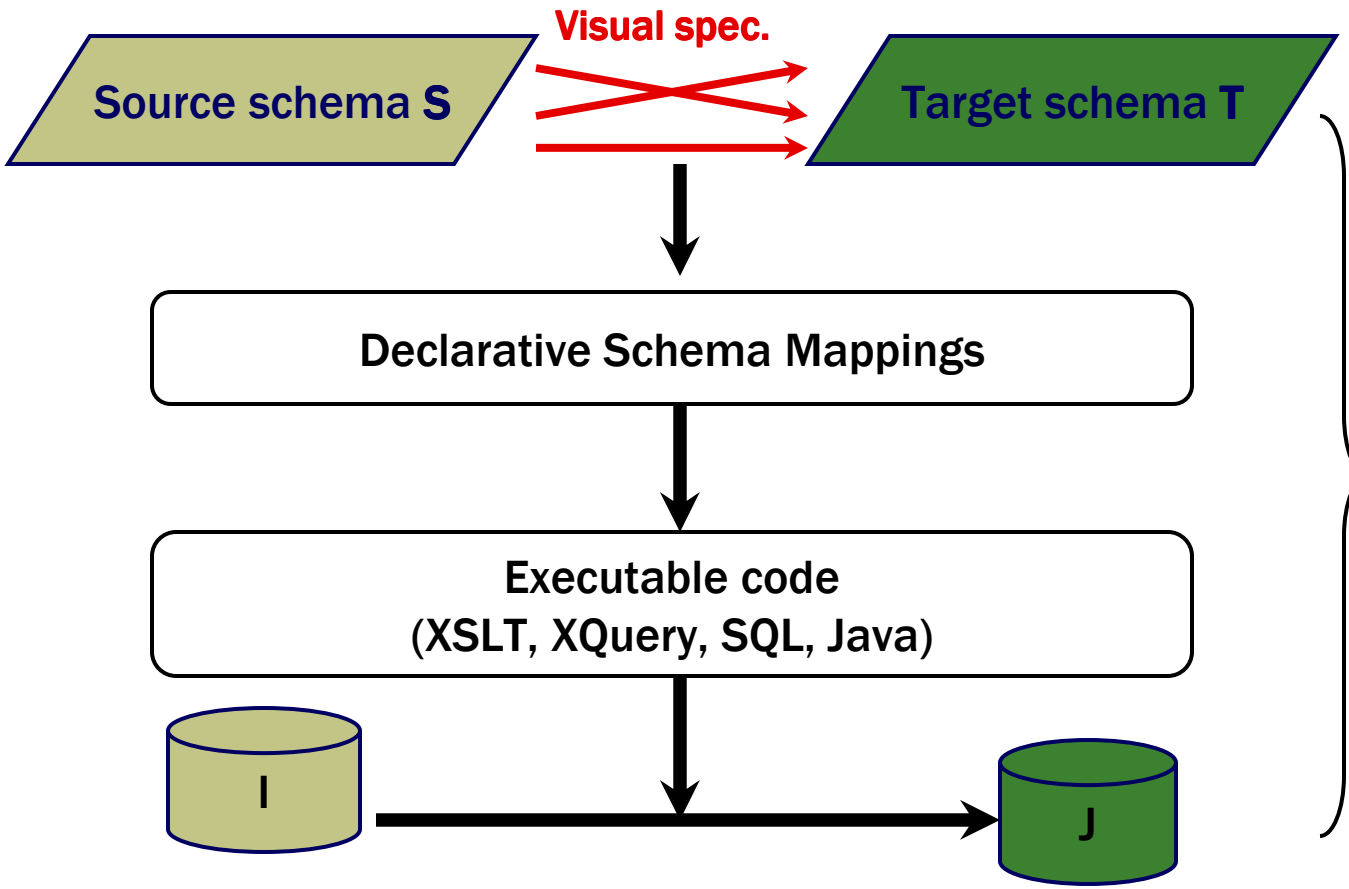
- Schema mappings:

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

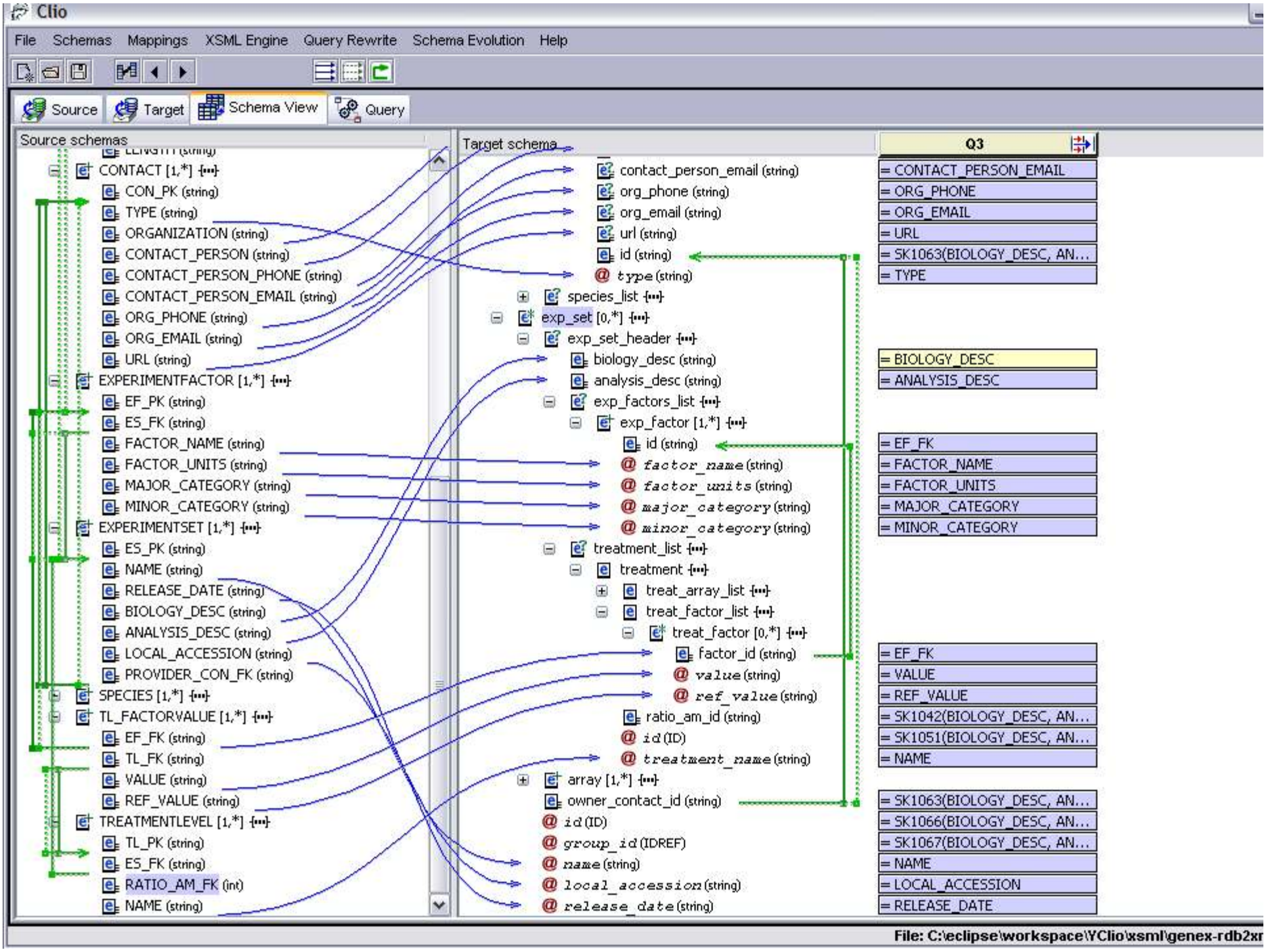
- Schema mappings constitute the essential building blocks in formalizing and studying data interoperability tasks, including data integration and data exchange.

- Schema mappings help with the development of tools.

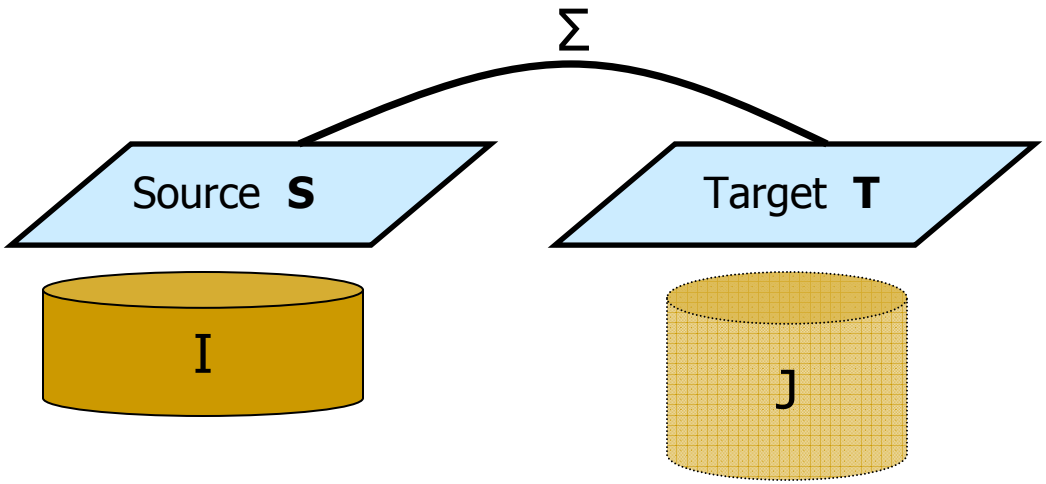
Schema-Mapping Systems: State-of-the-Art



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


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
Q3
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)
= 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

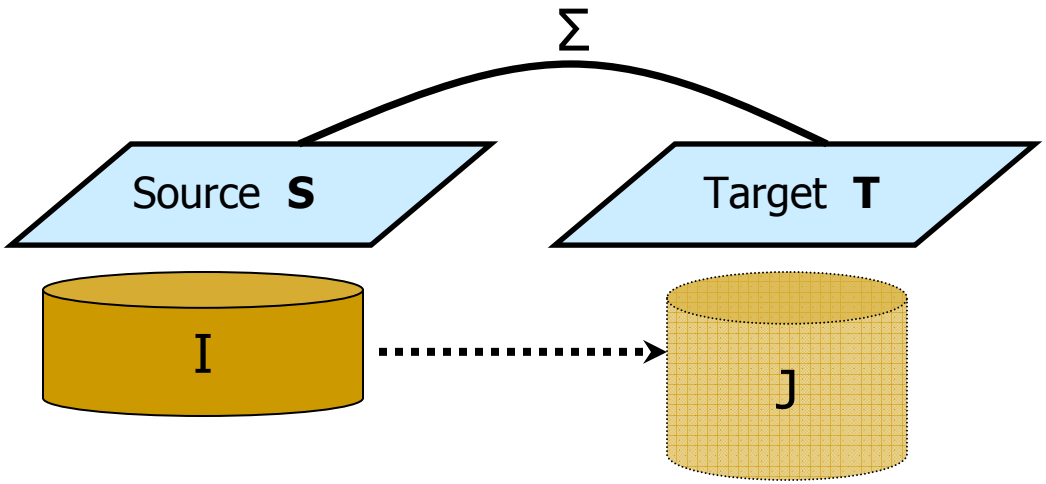
Schema Mappings



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

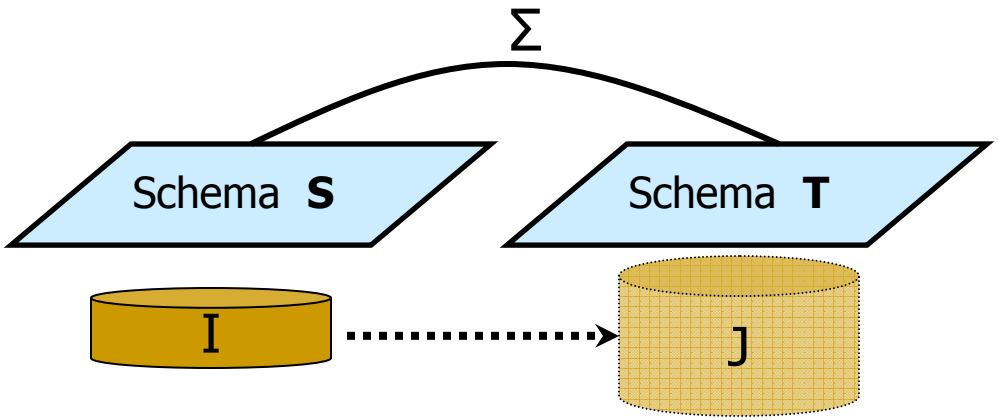
Fact: In general, for a given source instance I,

- No solution for I may exist (the constraints overspecify)

  or

- Multiple solutions for I may exist; in fact, infinitely many solutions for I may exist (the constraints underspecify).

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 a high-level, declarative language. In particular, it is natural to try to use relational calculus (first-order logic) as a specification language for schema mappings.
- **Fact**: There is a fixed relational calculus sentence specifying a schema mapping **M*** such that **Sol(M*)** is undecidable.
  - **Reason:** Undecidability of the Finite Validity Problem
- Hence, we need to restrict ourselves to well-behaved fragments of relational calculus.

Schema-Mapping Specification Languages: Bottom-Up

Let us consider some simple tasks that a schema mapping specification language should support:

- Copy (Nicknaming):
  - Copy each source table to a target table and rename it.
- Projection:
  - Form a target table by projecting on one or more columns of a source table.
- Decomposition:
  - Decompose a source table into two or more target tables.
- Column Augmentation:
  - Form a target table by adding one or more columns to a source table.
- Join:
  - Form a target table by joining two or more source tables.
- Combinations of the above (e.g., "join + column augmentation")

Schema Mapping Specification Languages

- Copy (Nicknaming):
  - $\forall x_1, \ldots, x_n (P(x_1,\ldots,x_n) \rightarrow R(x_1,\ldots,x_n))$
- Projection:
  - $\forall x,y,z (P(x,y,z) \rightarrow R(x,y))$
- Decomposition:
  - $\forall x,y,z\ (P(x,y,z) \rightarrow R(x,y) \wedge T(y,z))$
- Column Augmentation:
  - $\forall x,y\ (P(x,y) \rightarrow \exists\ z\ R(x,y,z))$
- Join:
  - $\forall x,y,z (E(x,z) \wedge F(z,y) \rightarrow R(x,y,z))$
- Combinations of the above (e.g., "join + column augmentation")
  - $\forall x,y,z (E(x,z) \wedge F(z,y) \rightarrow \exists\ w\ T(x,y,z,w)))$

Schema Mapping Specification Languages

- Question: What do all these tasks (copy, projection, decomposition, column augmentation, 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 formulas of relational calculus, called

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;
- $\mathbf{x}$ and $\mathbf{y}$ are tuples of variables.

They are also known as GLAV (Global-and-Local-as-View) constraints

Example:

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

(here, we have dropped the universal quantifiers in front of s-t tgds)

Schema Mapping Specification Language

- s-t tgds will be used to specify the relationship between source and target; they 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 (GLAV constrtaints) 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 relation.

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

    Note: Copy, projection, and join are GAV s-t tgds.

Target Dependencies

In addition to source-to-target dependencies, we also consider target dependencies, since, after all, the target schema may have its own integrity constraints:

- 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)

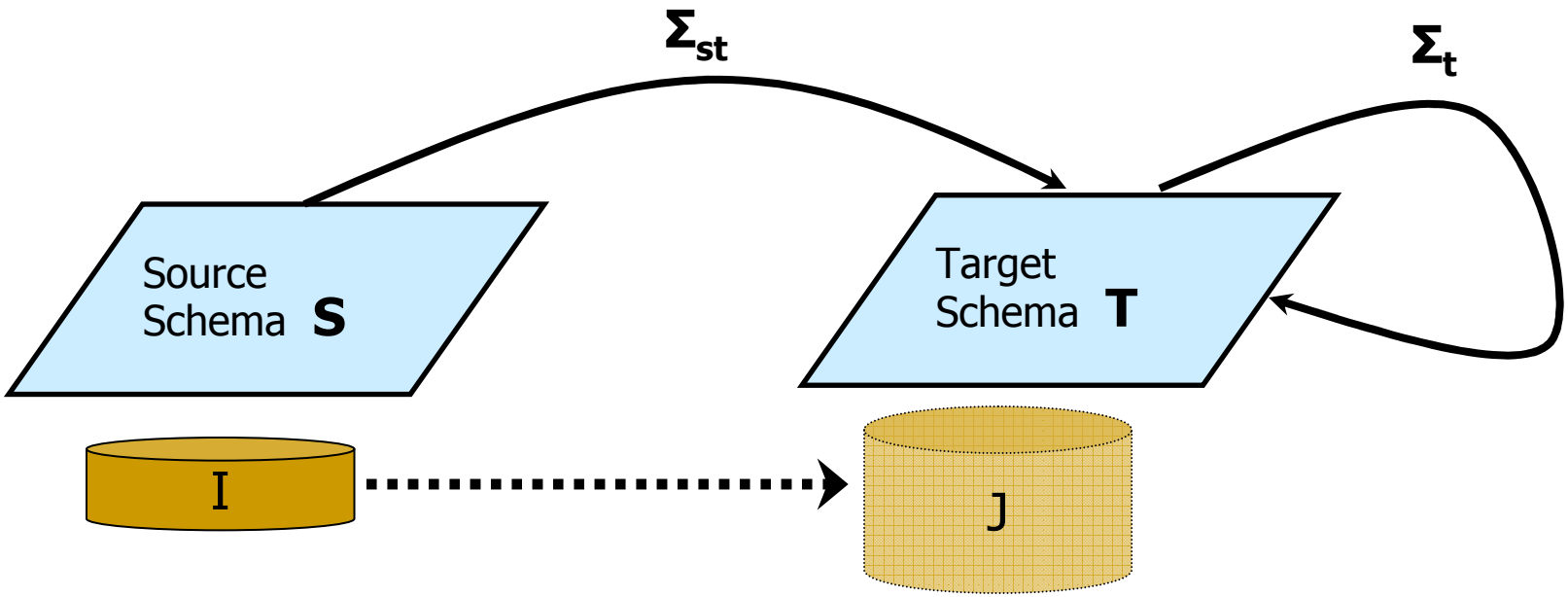
- Target Equality Generating Dependencies (egds):

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

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

  (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

Schema Mappings: An Example

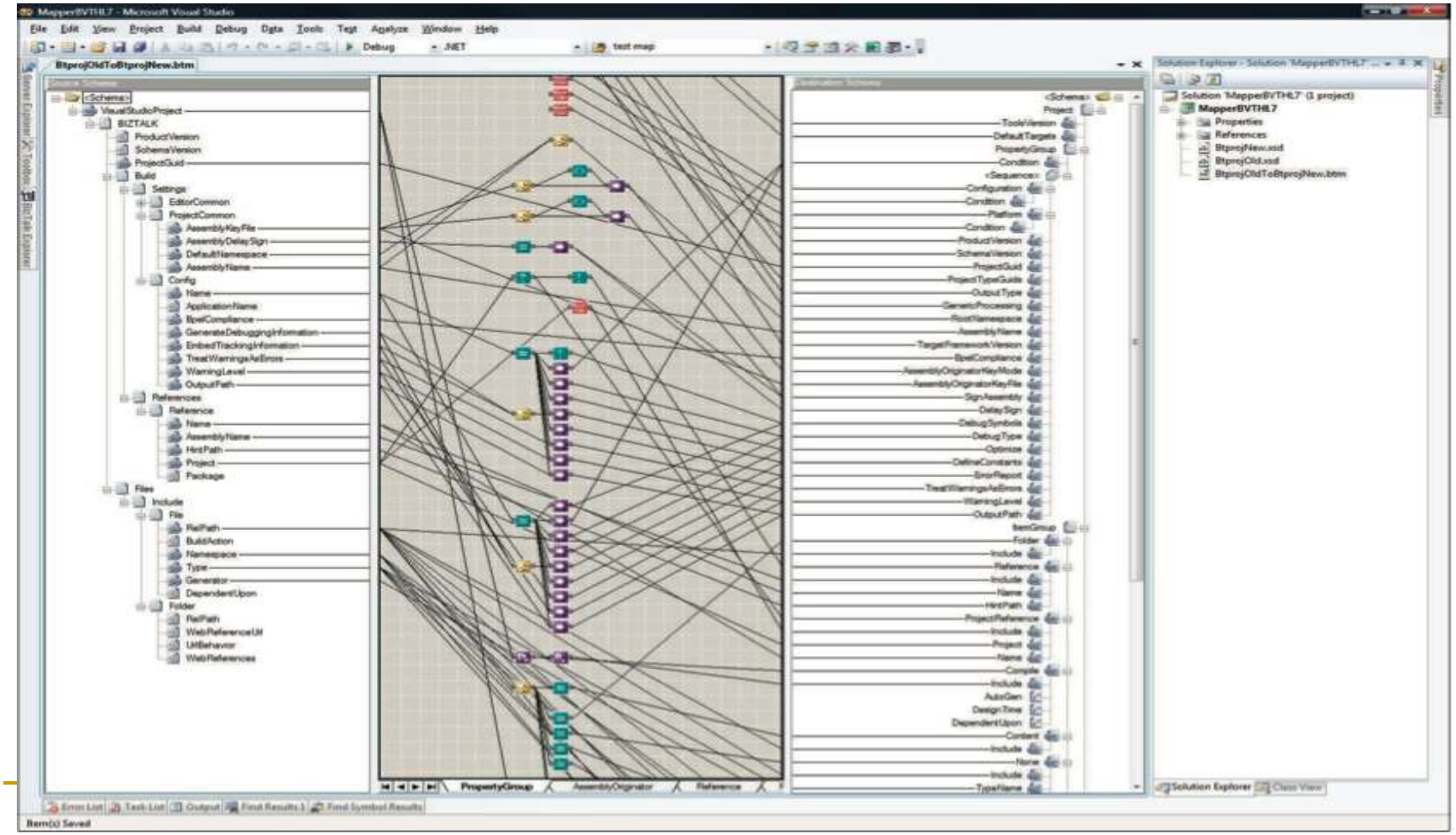
- Source Schema **S**: Movies database with relations
  P(title, year), R(title, director)
- Target Schema **T**: Movies database with relations
  Movies(title, year, director), Reviews(title, year, critic, score)

- Σ_{st} consists of the following source-to-target tgds
  - $\forall t\, \forall y\, \forall d\, (P(t,y) \wedge R(t,d) \rightarrow Movies(t,y,d))$ (GAV)
  - $\forall t\, \forall y\, (P(t,y) \rightarrow \exists c\, \exists s\, Reviews(t,y,c,s))$ (LAV)

- Σ_t consists of the following target tgds and target egds
  - $\forall t\, \forall y\, \forall d\, \forall d'\, (Movies(t,y,d) \wedge Movies(t,y,d') \rightarrow d = d')$
  - $\forall t\, \forall y\, \forall c\, \forall s\, \forall s'\, (Reviews(t,y,c,s) \wedge Reviews(t,y,c,s') \rightarrow s = s')$
  - $\forall t\, \forall y\, \forall c\, \forall s\, (Reviews(t,y,c,s) \rightarrow \exists d\, Movies(t,y,d))$

Visual Specification

- Screenshot from Bernstein and Haas 2008 CACM article.
  "*Information Integration in the Enterprise*"



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 ),(
```

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?

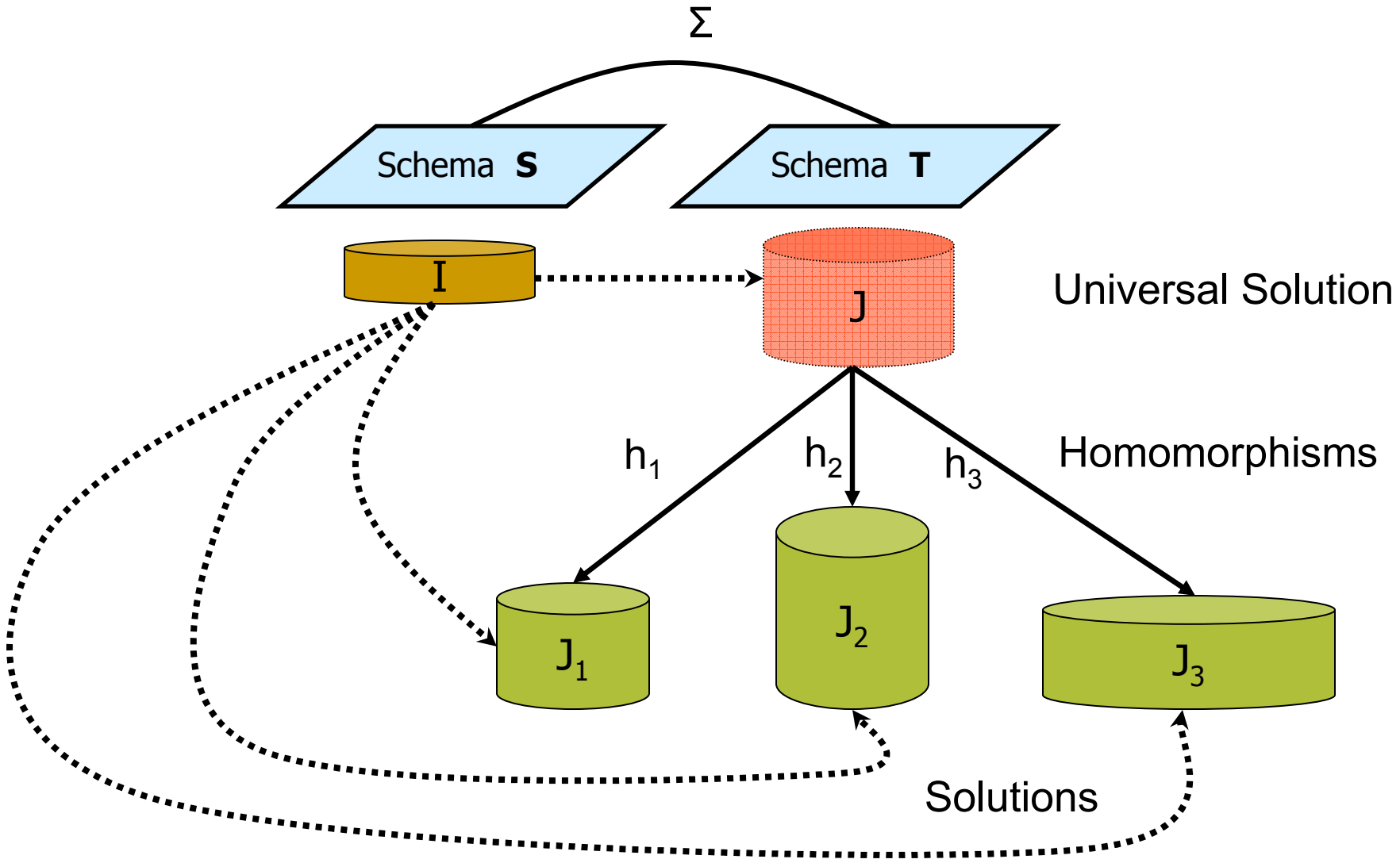
Universal Solutions in Data Exchange

Definition (Fagin, K …, Miller, Popa 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 = {E(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: Let **M** = (**S**, **T**, Σ_{st}) be a schema mapping with no target constraints, i.e., Σ_{st} is a set of s-t tgds and $\Sigma_t = \emptyset$. Then

- Solutions always exist; hence, **Sol(M)** is trivial.
- Universal solutions can be computed in polynomial time via the naïve chase procedure.

The Naïve Chase Algorithm

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}) (here $\Sigma_t = \emptyset$)

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

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

- If E= { (1,2) }, then F*={ (1,N),(N,2) } is universal solution for E
- If E ={ (1,2),(2,3),(1,4) }, then
- F*={ (1,M),(M,2),(2,N),(N,3),(1,U),(U,4) } is universal solution for E.

The Naïve Chase Algorithm

Example : Collapsing paths of length 2 to edges

M = (**S**, **T**, Σ_{st}) (here $\Sigma_t = \emptyset$)

Σ_{st}: $E(x,z) \wedge E(z,y) \rightarrow F(x,y)$ (GAV mapping)

- E = { (1,3), (2,4), (3,4) }
  F* = { (1,4) } Universal Solution for E

- E = { (1,3}, (2,4), (3,4), (4,3) }
  F* = { (1,4), (2,3), (3,3), (4,4) } Universal solution for E

Algorithmic Problems in Data Exchange

Question:

What about arbitrary target tgds and egds?

More formally:

What can we say about the existence-of-solutions problem for schema mappings **M**= (**S**, **T**, Σ^*_{st}, Σ^*_t) such that

- Σ^*_{st} is a set of s-t tgds;
- Σ^*_t is a set of target tgds and target egds?

The Complexity of the Existence of Solutions Problem

M = (**S**, **T**, Σ_{st} , Σ_t) Σ_{st} a set of s-t tgds	Existence-of-Solutions Problem	Existence-of-Universal Solutions Problem	Computing a Universal Solution
$\Sigma_t = \varnothing$ No target constraints	Trivial	Trivial	PTIME
Σ_t: Weakly acyclic set of target tgds + egds	PTIME It can be PTIME-complete	PTIME Univ. solutions exist if and only if solutions exist	PTIME
Σ_t: target tgds + egds	Undecidable, in general	Undecidable, in general	No algorithm exists, in general

感谢您