Deductive Databases

Data Exchange

2020/2021

Outline

Introduction

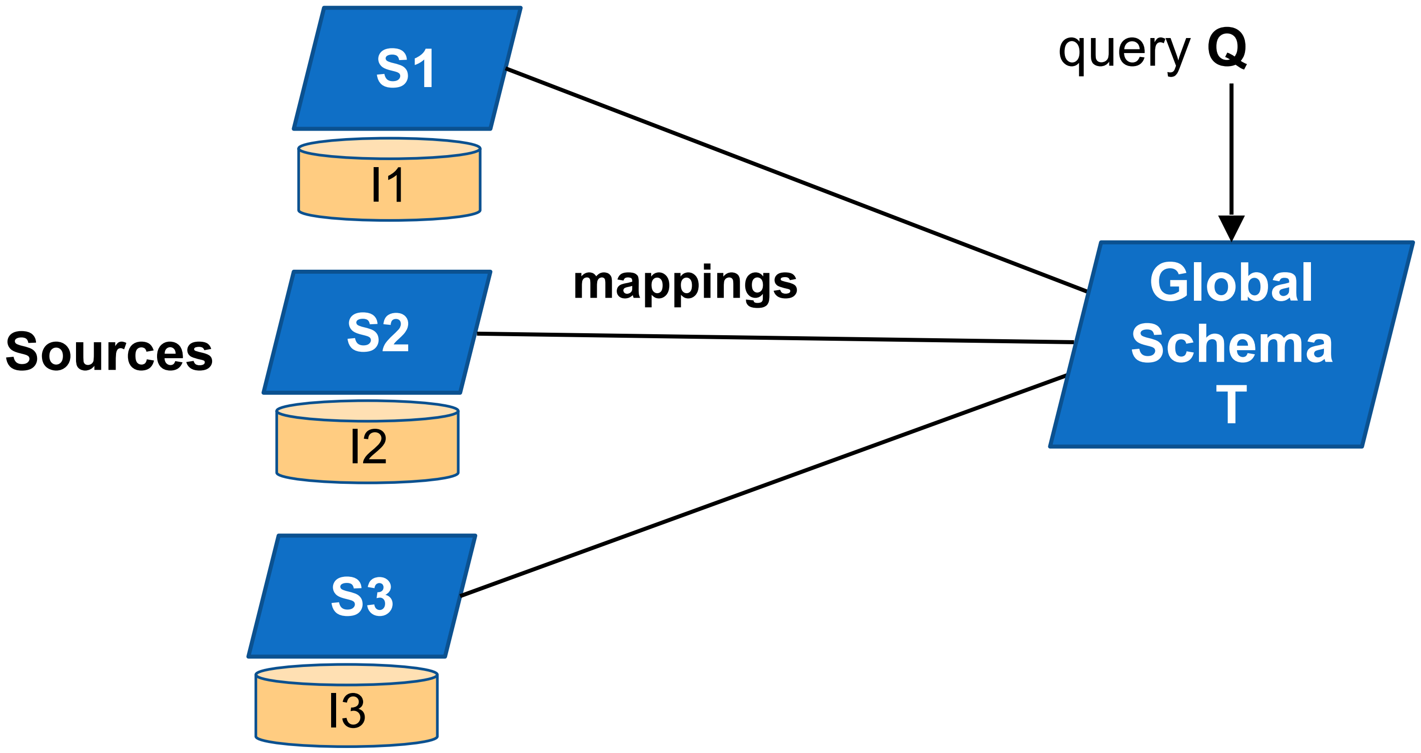
- Data is inherently heterogeneous
  - Due to the explosion of online data repositories
  - Due to the variety of users, who develop a wealth of applications
    - at different time
    - with disparate requirements in their mind

Introduction

- A fundamental requirement is to translate data across different formats and to ensure data interoperability
  - Two facets of the same problem
    - Data Integration
    - Data Exchange
  - Other important related problems
    - Schema Integration
    - Schema Evolution

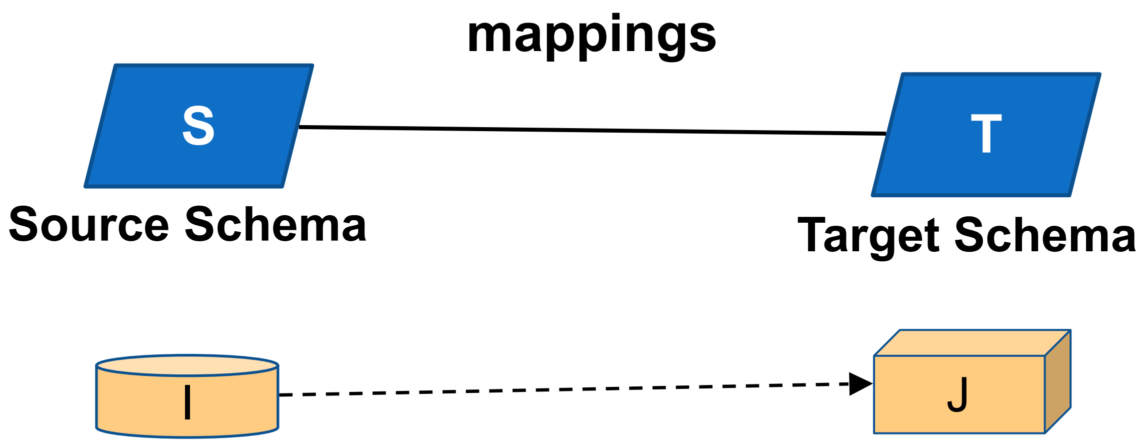
Data Integration

- Data Integration [Lenzerini 2002]
  - Query heterogeneous data in different sources via a virtual global schema



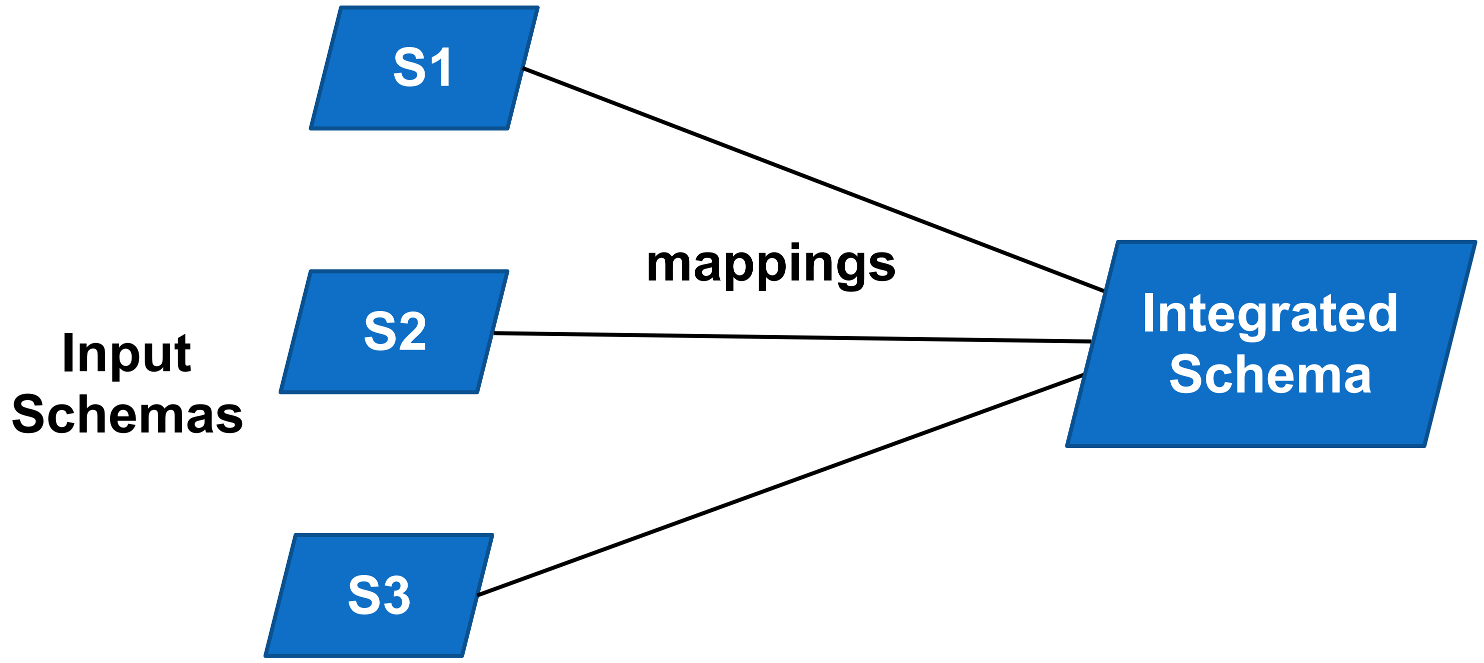
Data Exchange

- Data Exchange [Fagin et al. 2005]
  - Transform data structured under a source schema into data structured under a different target schema



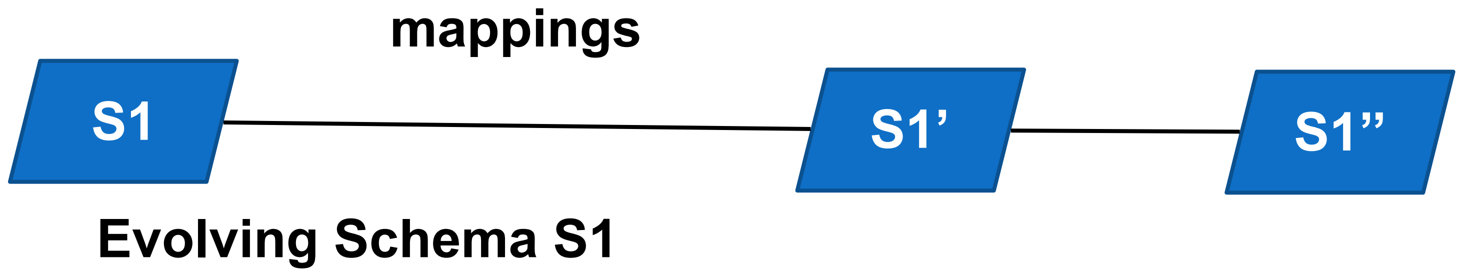
Schema Integration

- Schema Integration [Batini et al. 1986]
  - A set of source schemas need to be integrated into one mediated schema



Schema Evolution

- Schema Evolution [Lerner 2000]
  - An original schema S1 evolves into subsequent versions S1', S1'' etc.



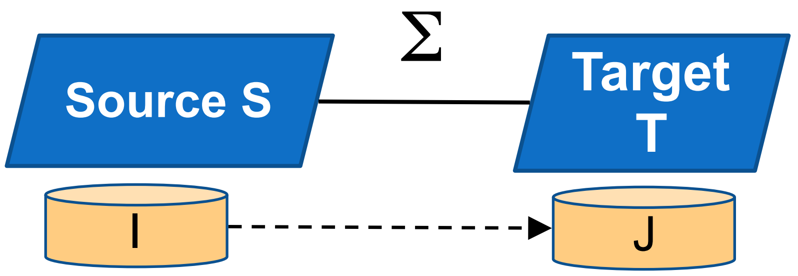
Data Exchange

- Data Exchange is an old, but recurrent, database problem
- Phil Bernstein, Microsoft – 2003 “Data exchange is the oldest database problem”
  - EXPRESS: IBM San Jose Research Lab – 1977
    EXtraction, Processing, and REStructuring System for transforming data between hierarchical databases.
- Data Exchange underlies: Data Warehousing, ETL (Extract-Transform-Load) tasks; XML Publishing, XML Storage; more recently, exporting relational data to RDF.

Schema Mapping

- Schema mappings: high-level, declarative assertions that specify the relationship between two schemas
- Ideally, schema mappings should be
  - Expressive enough to specify data interoperability tasks
  - Simple enough to be efficiently manipulated by tools
- Schema mappings constitute the essential building blocks in formalizing data integration and data exchange
- Schema mapping help with the development of tools:
  - are easier to generate and manage (semi)-automatically;
  - can be compiled into SQL/XSLT scripts automatically

Schema Mapping



- Schema Mapping $M = (S, T, \Sigma)$
  - 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 $M = (S, T, \Sigma)$
  - Transform a given source instance I to a target instance J, so that $<I, J>$ satisfy the specifications Σ of M

Solutions in schma mappings

- **Definition**: Schema Mapping $M = (S, T, \Sigma)$ 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 Mapping specification languanges

- **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 exists a fixed first-order sentence specifying a schema mapping M^* such that $Sol(M^*)$ is undecidable
  - Reason: undecidability of validity in FOL
  - Hence, we need to restrict ourselves to well-behaved fragments of first-order logic

Embedded Dependencies

- **Dependency Theory:** extensive study of constraints in relational databases in the 1970s and 1980s
- **Embedded Implicational Dependencies:** R. Fagin, C. Beeri and M. Vardi, …
  - Class of constraints with a balance between high expressive power and good algorithmic properties
  - **Tuple-generating dependencies (tgds):**
    - Inclusion and multi-valued dependencies are a special case
  - **Equality-generating dependencies (egds):**
    - Functional dependencies are a special case

Schema Mapping specification languanges

- How the relationship between source and target is given by formulas of first-order logic,
  - Source-to-Target Tuple Generating Dependencies (s-t tgds)
- $\varphi(x) \rightarrow \exists y\, \Psi(x, y)$
  - $\varphi(x)$ is a conjunction of atoms over the source;
  - $\Psi(x, y)$ is a conjunction of atoms over the target
- Example
  - $\left(\text{Student}(s) \wedge \text{Enrolls}(s, c)\right) \rightarrow \exists\, t\, \exists\, g\, \left(\text{Teachers}(t, c) \wedge \text{Grades}(s, c, g)\right)$

Examples of simple mapping tasks

- 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
  - …

Examples of simple mapping tasks

- Let us consider some simple tasks that a schema mapping specification language should support:
  - …
  - `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
  - Combination of the above
    - e.g., “`join + column augmentation`”

Examples of simple mapping tasks

- `Copy(Nicknaming):`
  - Copy each source table to a target table and rename it
  - $\forall x_1, \ldots, x_n \left(P(x_1, \ldots, x_n) \rightarrow R(x_1, \ldots, x_n)\right)$
- `Projection:`
  - Form a target table by projecting on one or more columns of a source table
  - $\forall x, y, z \left(P(x, y, z) \rightarrow R(x, y)\right)$
- `Decomposition:`
  - Decompose a source table into two or more target tables
  - $\forall x, y, z \left(P(x, y, z) \rightarrow \left(R(x, y) \wedge T(y, z)\right)\right)$

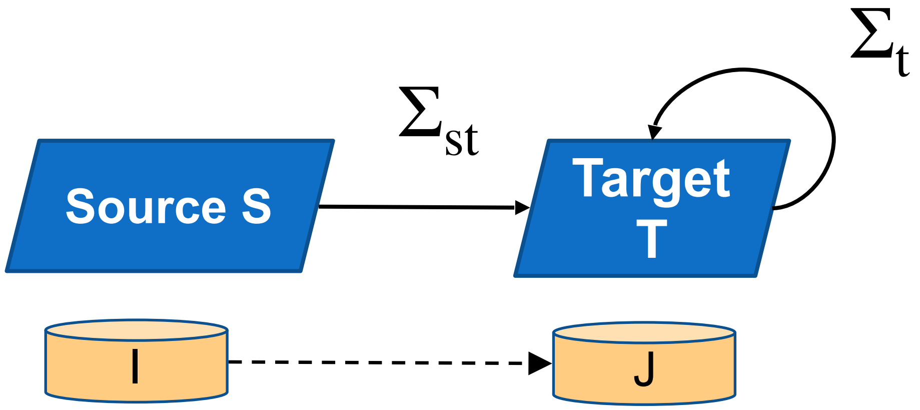
Examples of simple mapping tasks

- Column Augmentation:
  - Form a target table by adding one or more columns to a source table
  - $\forall x, y \left(P(x,y) \rightarrow \exists z\, R(x,y,z)\right)$
- Join:
  - Form a target table by joining two or more source tables
  - $\forall x,\ y, z \left(E(x,z) \wedge F(z,y) \rightarrow R(x,y,z)\right)$
- join + column augmentation:
  - Combination of the above
  - $\forall x,\ y, z \left(E(x,z) \wedge F(z,y) \rightarrow \exists w\, T(x,y,z,w)\right)$

Target Dependencies

- In addition to source-to-target dependencies, we also consider target dependencies
  - **Target Tgds:** $\varphi^T(x) \rightarrow \exists y\ \Psi^T(x, y)$
  - An example of target inclusion dependency constraint
    - `Dept(did,dname,mgr_id,mgr_name)→Mgr(mgr_id,did)`
  - **Target Egds:** $\varphi^T(x) \rightarrow (x1 = x2)$
  - An example of target key constraint
    - `(Mgr(e,d1)∧ Mgr(e,d2))→ (d1=d2)`

Data Exchange Framework



- <span>Schema Mapping</span> $M = (S, T, \Sigma_{st}, \Sigma_t)$
  - Σ_{st} is a set of source-to-target tgds
  - Σ_t is a set of target tgds and target egds

Multiple Solutions

- **Fact:** Given a source instance, multiple solutions may exist
- **Example:**
  - Source relation $E(A,B)$
  - target relation $H(A,B)$

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

  - Source instance:
    - $I = \{E(a,b)\}$
  - Solutions:
    - Infinitely many solutions exist

Multiple Solutions: Example

- Consider a set of source-to-target dependencies Σ_{st}:
  - (d1) `EmpCity(e,c)` → $\boxed{?}$`H Home(e,H)`
  - (d2) `EmpCity(e,c)` → $\boxed{?}$`D (EmpDept(e,D)` ∧ `DepCity(D,c))`
  - (d3) `LivesIn(e,h)` → `Home(e,h)`
  - (d4) `LivesIn(e,h)` → ∃ `D` ∃`C(EmpDept(e,D)` ∧ `DepCity(D,C))`
- and a source instance I such that:

  -
    ```
    I = { EmpCity(Alice,SJ)
          EmpCity(Bob,SD)
          LivesIn(Alice,SF)
          LivesIn(Bob,LA) }
    ```
- *Which possible solution do exist?*

Multiple Solutions: Example

A possible solution:

```
J0 = { Home(Alice,SF),
       Home(Bob,LA),
       EmpDept(Alice,D1),
       EmpDept(Bob,D2),
       DepCity(D1,SJ),
       DepCity(D2,SD) }
```

I: `LivesIn(Alice,SF)`
`LivesIn(Bob,LA)`
(d3): `LivesIn(e,h)` → `Home(e,h)`

I: `EmpCity(Alice,SJ)`
`EmpCity(Bob,SD)`
(d2): `EmpCity(e,c)` → $\boxed{?}$ `D(EmpDept(e,D)` ∧
`DepCity(D,c))`

Multiple Solutions: Example

A possible solution:

```
J0’= { Home(Alice,SF),
       Home(Bob,LA),
       EmpDept(Alice,D),
       EmpDept(Bob,D),
       DepCity(D,SJ),
       DepCity(D,SD)  }
```

```
  I:  LivesIn(Alice,SF)
      LivesIn(Bob,LA)
(d3): LivesIn(e,h)→ Home(e,h)
```

```
  I:  EmpCity(Alice,SJ)
      EmpCity(Bob,SD)
(d2): EmpCity(e,c)→
                    ? D(EmpDept(e,D)∧
DepCity(D,c))
```

Multiple Solutions: Example

A possible solution:

```
J  = { Home(Alice,SF),
       Home(Bob,LA),
       Home(Alice,H1),
       Home(Bob,H2),
       EmpDept(Alice,D1),
       EmpDept(Bob,D2),
       DepCity(D1,SJ),
       DepCity(D2,SD) }
```

I: LivesIn(Alice,SF)
LivesIn(Bob,LA)
(d3): LivesIn(e,h)→ Home(e,h)

I: EmpCity(Alice,SJ)
EmpCity(Bob,SD)
(d1): EmpCity(e,c)→ $\boxed{?}$ H
Home(e,H)

I: EmpCity(Alice,SJ)
EmpCity(Bob,SD)
(d2): EmpCity(e,c)→
$\boxed{?}$ D(EmpDept(e,D) ∧
DepCity(D,c))

Main issues in Data Exchange

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

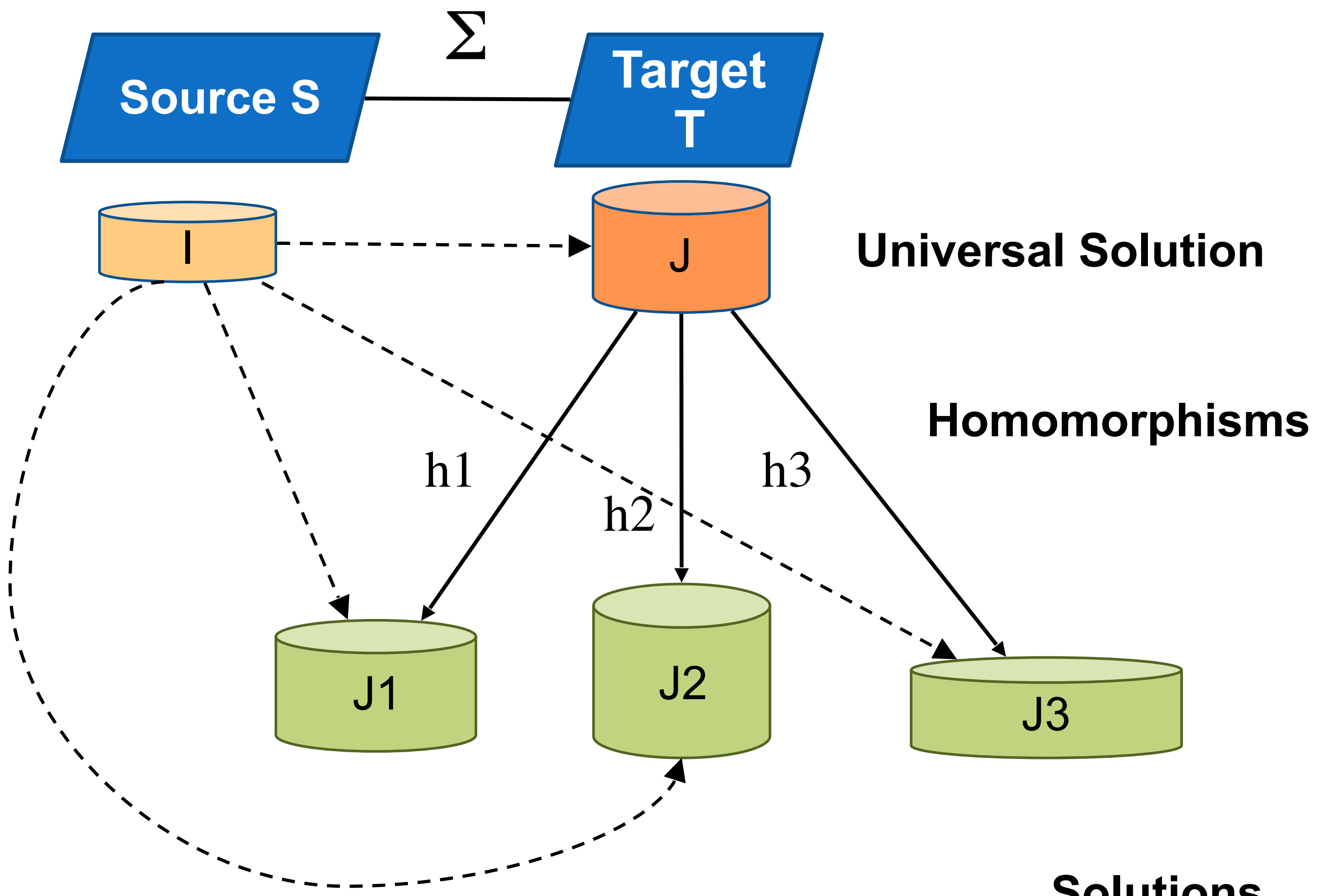
Universal solution in Data Exchange

- We introduce the notion of universal solutions as the “best” solutions in data exchange
- By definition, a solution is universal if it has homomorphisms to all other solutions
  - It is a “most general” solution
- Constants
  - entries in source instances
- Variables (labeled nulls)
  - invented entries in target instances

Universal solution in Data Exchange

- By definition, a solution is universal if it has homomorphisms to all other solutions
  - It is a “most general” solution
- Homomorphism $\mathrm{h}:\mathrm{J1} \rightarrow \mathrm{J2}$ between target instances:
  - $\mathrm{h}(\mathrm{c}) = \mathrm{c}$, for every constant c in $\mathrm{J1}$
  - For every fact $\mathrm{P}\left(a_1,\ \ldots, a_m\right)$ in $\mathrm{J1}$, then we have that $\mathrm{P}\left(\mathrm{h}(a_1),\ \ldots, \mathrm{h}(a_m)\right)$ is a fact in $\mathrm{J2}$

Schema Mapping



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:
  - I and I' have the same space of solutions.
  - J and J' are homomorphically equivalent.

Weakly Acyclic Sets of Tgds: Definition

- Position graph (dependency 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 occurrence of *x* in φ in position R.A
    - for every *x* in x occurring in Ψ in position S.B
      - add an edge R.A $\rightarrow$ S.B (if it does not already exist)
    - for every existentially quantified variable *y* in Ψ and for every occurrence *y* in Ψ in position T.C
      - add a special edge R.A ➡ T.C (if it does not already exist)
- Σ is weakly acyclic if the position graph has **no** cycle containing a **special edge**

Weakly Acyclic Sets of Tgds

- Weakly acyclic sets of tgds contain as special cases
  - Sets of full tgds

$\varphi^T(x, x') \rightarrow \Psi^T(x)$

  - $\varphi^T(x, x')$ and $\Psi^T(x)$ are conjunctions of atoms
- Acyclic sets of inclusion dependencies
  - Large class of dependencies occurring in practice

Weakly Acyclic Sets of Tgds: Examples

- **Example 1:** `D(d1,d2)` and `M(m1)`
  - $\Sigma = \{D(e,m) \rightarrow M(m),\ M(m) \rightarrow \exists e\, D(e,m)\}$

$$\begin{tikzcd}
D.d1 & M.m1 \arrow[l, Rightarrow] \arrow[r, shift right] & D.d2 \arrow[l, shift right]
\end{tikzcd}$$

is weakly acyclic

- **Example 2:** `E(e1,e2)`
  - $\Sigma = \{E(x,y) \rightarrow \exists z\, E(y,z)\}$

$$\begin{tikzcd}
E.e1 & E.e2 \arrow[l] \arrow[loop right, Rightarrow]
\end{tikzcd}$$

is not weakly acyclic