Schema Mappings and Data Examples

Balder ten Cate
UC Santa Cruz & LogicBlox

TbiLLC 2013 - Gudauri

Relational Databases for Logicians*

*) an oversimplified picture.

- Database schema ~ a finite relational signature. E.g.,
  - { PARTICIPANT(name, email, flight-nr), FLIGHT(flight-nr, dept-time) }

Relational Databases for Logicians*

*) an oversimplified picture.

- Database schema ~ a finite relational signature. E.g.,
  - { PARTICIPANT(name, email, flight-nr), FLIGHT(flight-nr, dept-time) }
- Database instance (of a given schema) ~ a finite structure.

Relational Databases for Logicians*

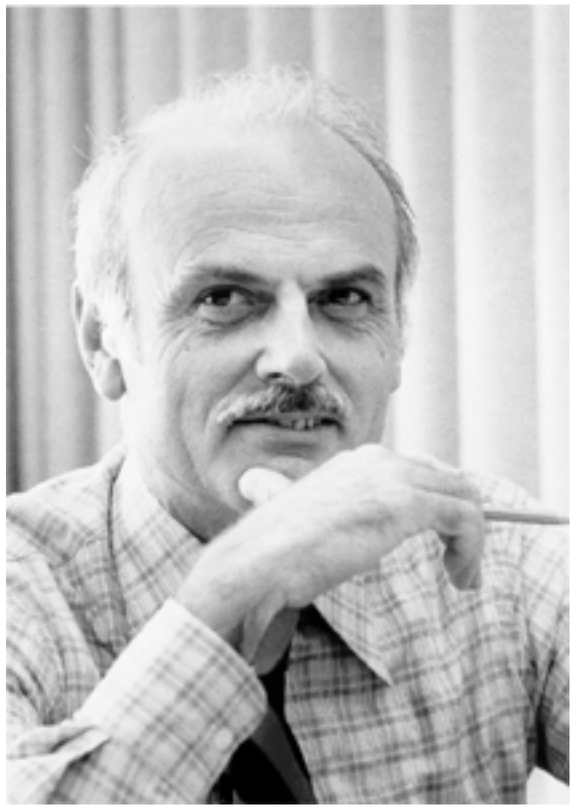
*) an oversimplified picture.

- Database schema ~ a finite relational signature. E.g.,
  - { PARTICIPANT(name, email, flight-nr), FLIGHT(flight-nr, dept-time) }
- Database instance (of a given schema) ~ a finite structure.
- Database queries ~ logical formulas with free variables
  - $\phi(x,y) = \exists z,u\ (\mathrm{PARTICIPANT}(x,y,z)\ \&\ \mathrm{FLIGHT}(z, 3{:}00\mathrm{AM}))$

Relational Databases for Logicians*

*) an oversimplified picture.

- Database schema ~ a finite relational signature. E.g.,
  - { PARTICIPANT(name, email, flight-nr), FLIGHT(flight-nr, dept-time) }
- Database instance (of a given schema) ~ a finite structure.
- Database queries ~ logical formulas with free variables
  - $\phi(x,y) = \exists z,u\ (\mathrm{PARTICIPANT}(x,y,z)\ \&\ \mathrm{FLIGHT}(z, 3{:}00\mathrm{AM}))$
- Database constraints ~ logical sentences expressing structural properties
  - $\forall x,y,z,u\ (\mathrm{PARTICIPANT}(x,y,z) \rightarrow \exists t\ \mathrm{FLIGHT}(z,t))$
  - $\forall x,y,z\ (\mathrm{FLIGHT}(x,y)\ \&\ \mathrm{FLIGHT}(x,z) \rightarrow y=z)$



Edgar F. Codd (1923-2003)

Query Languages

- Most important query languages
  - Conjunctive Queries (CQs): $\psi(\mathbf{x}) = \exists \mathbf{y}\, (\alpha_1(\mathbf{x},\mathbf{y}) \wedge \ldots \wedge \alpha_n(\mathbf{x},\mathbf{y}))$
  - Unions of Conjunctive Queries (UCQs): disjunctions of CQs.
  - First-order Queries (~ SQL queries)
  - Datalog (the least-fixpoint extension of UCQs)

Query Languages

- Most important query languages
  - Conjunctive Queries (CQs): $\psi(\mathbf{x}) = \exists \mathbf{y}\ (\alpha_1(\mathbf{x},\mathbf{y}) \wedge \ldots \wedge \alpha_n(\mathbf{x},\mathbf{y}))$
  - Unions of Conjunctive Queries (UCQs): disjunctions of CQs.
  - First-order Queries (~ SQL queries)
  - Datalog (the least-fixpoint extension of UCQs)
- Most database queries in practice are CQs (a.k.a. SELECT-FROM-WHERE)

Query Languages

- Most important query languages
  - Conjunctive Queries (CQs): $\psi(\mathbf{x}) = \exists \mathbf{y}\, (\alpha_1(\mathbf{x},\mathbf{y}) \wedge \ldots \wedge \alpha_n(\mathbf{x},\mathbf{y}))$
  - Unions of Conjunctive Queries (UCQs): disjunctions of CQs.
  - First-order Queries (~ SQL queries)
  - Datalog (the least-fixpoint extension of UCQs)
- Most database queries in practice are CQs (a.k.a. SELECT-FROM-WHERE)
- UCQs form a "robustly decidable" fragment of FO logic.
  - In particular, equivalence is decidable (NP-complete).

Excursion: decidable fragments of FO

- Unions of Conjunctive queries:
  - $\phi(\mathbf{x}) := R(\mathbf{x}) \mid x_i = x_j \mid \phi(x) \wedge \phi(x) \mid \phi(x) \vee \phi(x) \mid \exists y\, \phi(\mathbf{x}, y)$
- The modal fragment:
  - $\phi(x) := P(x) \mid \phi(x) \wedge \phi(x) \mid \neg\phi(x) \mid \exists y (Rxy \wedge \phi(y))$

Excursion: decidable fragments of FO

- Unions of Conjunctive queries:
  - $\phi(\mathbf{x})$:= $R(\mathbf{x})$ | $x_i = x_j$ | $\phi(x) \wedge \phi(x)$ | $\phi(x) \vee \phi(x)$ | $\exists y\, \phi(\mathbf{x}, y)$
- The modal fragment:
  - $\phi(x)$:= $P(x)$ | $\phi(x) \wedge \phi(x)$ | $\neg\phi(x)$ | $\exists y(Rxy \wedge \phi(y))$
- UNFO (Unary-Negation Fragment of FO) [tC & Segoufin 2011]
  - $\phi(\mathbf{x})$:= $R(\mathbf{x})$ | $x_i = x_j$ | $\phi(\mathbf{x}) \wedge \phi(\mathbf{x})$ | $\phi(\mathbf{x}) \vee \phi(\mathbf{x})$ | $\neg\phi(x)$ | $\exists y\, \phi(\mathbf{x}, y)$

Excursion: decidable fragments of FO

- Unions of Conjunctive queries:
  - $\phi(\mathbf{x}) := R(\mathbf{x}) \mid x_i = x_j \mid \phi(x) \wedge \phi(x) \mid \phi(x) \vee \phi(x) \mid \exists y\, \phi(\mathbf{x}, y)$
- The modal fragment:
  - $\phi(x) := P(x) \mid \phi(x) \wedge \phi(x) \mid \neg\phi(x) \mid \exists y (Rxy \wedge \phi(y))$
- UNFO (Unary-Negation Fragment of FO) [tC & Segoufin 2011]
  - $\phi(\mathbf{x}) := R(\mathbf{x}) \mid x_i = x_j \mid \phi(\mathbf{x}) \wedge \phi(\mathbf{x}) \mid \phi(\mathbf{x}) \vee \phi(\mathbf{x}) \mid \neg\phi(x) \mid \exists y\, \phi(\mathbf{x}, y)$
- Further extension: GNFO (Guarded-Negation Fragment of FO) [Barany, tC & Segoufin 2011]

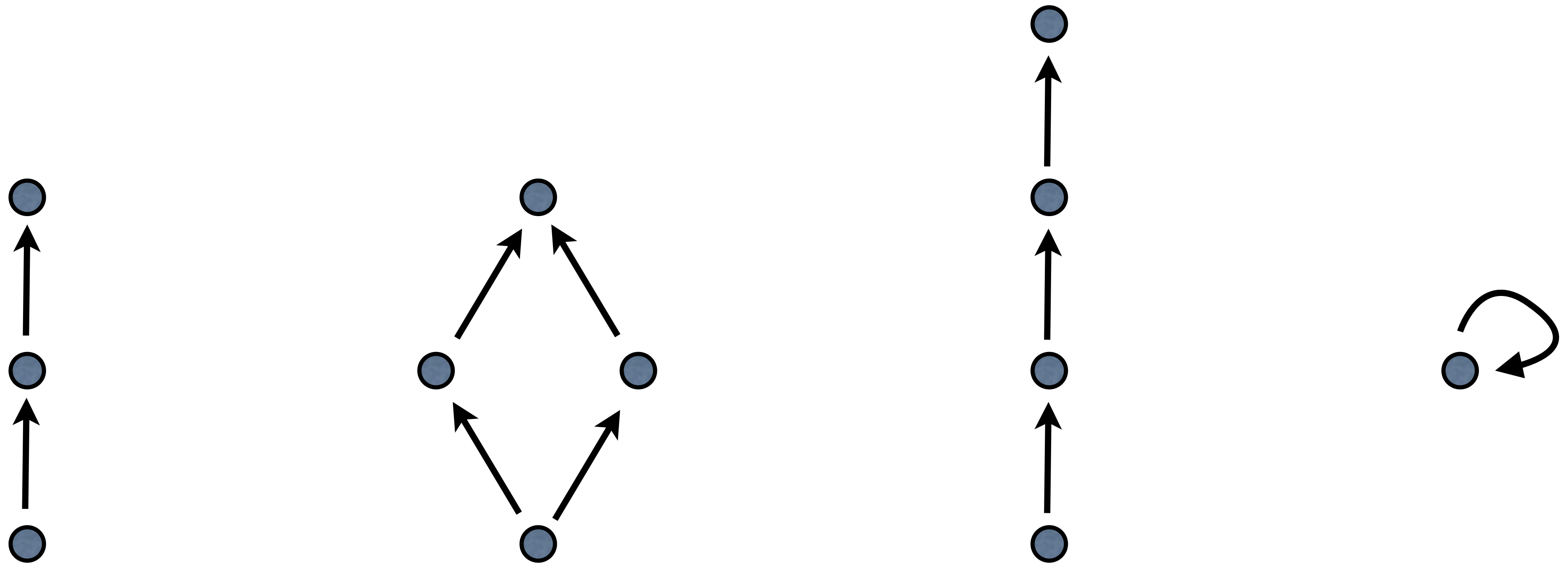
Homomorphisms

- Conjunctive queries are intimately tied to homomorphisms.

Homomorphisms

- Conjunctive queries are intimately tied to homomorphisms.
- **Definition**:
  - Let I and J be instances (i.e., finite structures) over the same schema. A homomorphism h: I $\rightarrow$ J is a map from the domain of I to the domain of J such that $(a,b,c) \in R^I$ implies $(h(a),h(b),h(c)) \in R^J$.

Examples

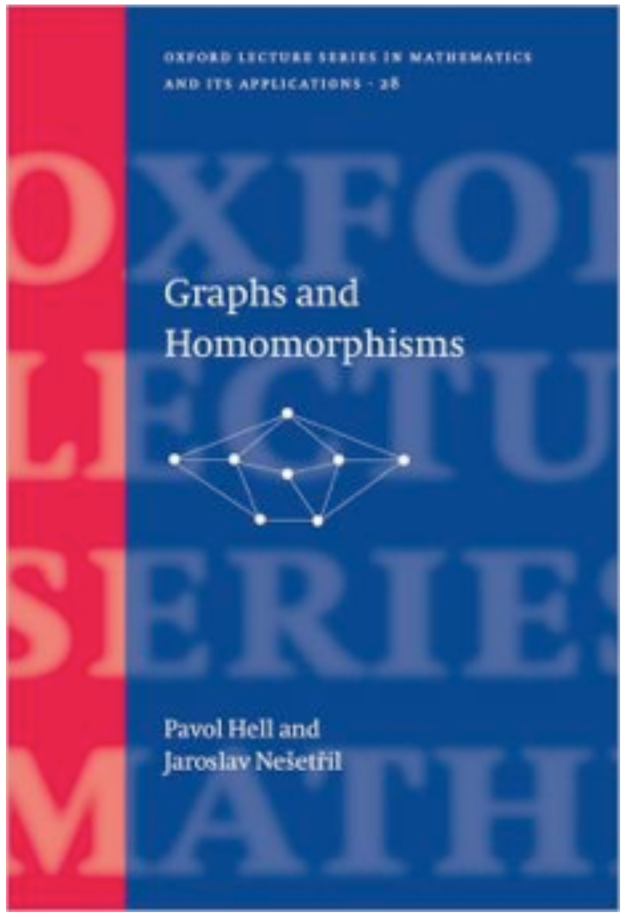


- **Def:** A query q is preserved by homomorphism if for all instances I and J and for all homomorphisms h:I → J, (a,b,c) ∈ q(I) implies (h(a),h(b),h(c)) ∈ q(J).

- **Def:** A query q is preserved by homomorphism if for all instances I and J and for all homomorphisms h:I $\rightarrow$ J, (a,b,c) $\in$ q(I) implies (h(a),h(b),h(c)) $\in$ q(J).
- **Thm.** A first-order query is preserved by homomorphisms if and only if it is equivalent to a union of conjunctive queries [Rossman 2005].
  - One of the few preservation theorems that hold over finite structures.

The Homomorphism Quasi-Order

- We write I → J if there is a homomorphism h: I → J.
- Fix any relational schema S and let FinStr[S] be the finite structures (i.e., instances) over S.
- (FinStr[S], →) is a quasi-order (reflexive and transitive).
- Its structure has been extensively studied. We will make use of some beautiful results from this area.


OXFORD LECTURE SERIES IN MATHEMATICS AND ITS APPLICATIONS · 28
Graphs and Homomorphisms
Pavol Hell and Jaroslav Nešetřil

Database Constraints

- Database constraints express structural properties of relations in a schema.
  - $\forall x,y,z,u$ (PARTICIPANT$(x,y,z) \rightarrow \exists t$ FLIGHT(z,t))
  - $\forall x,y,z$ (FLIGHT(x,y) & FLIGHT$(x,z) \rightarrow y=z$)

Database Constraints

- Database constraints express structural properties of relations in a schema.
  - $\forall x,y,z,u\ (\text{PARTICIPANT}(x,y,z) \rightarrow \exists t\ \text{FLIGHT}(z,t))$
  - $\forall x,y,z\ (\text{FLIGHT}(x,y)\ \&\ \text{FLIGHT}(x,z) \rightarrow y=z)$
- Traditional uses of constraints:
  - Schema design,   integrity control,   query optimization

Database Constraints

- Database constraints express structural properties of relations in a schema.
  - $\forall x,y,z,u$ (PARTICIPANT$(x,y,z) \rightarrow \exists t$ FLIGHT(z,t))
  - $\forall x,y,z$ (FLIGHT(x,y) & FLIGHT$(x,z) \rightarrow y=z$)
- Traditional uses of constraints:
  - Schema design, integrity control, query optimization
- The most well-studied language for specifying constraints:
  - Dependencies : $\forall \mathbf{x}\ (\alpha_1 \wedge \ldots \wedge \alpha_n \rightarrow \exists \mathbf{y}\ (\beta_1 \wedge \ldots \wedge \beta_n))$
  - Rich enough to express most database constraints in practice.
  - Unfortunately, basic tasks (e.g., entailment) are undecidable.

Relational Databases for Logicians*

*) an oversimplified picture.

- Database schema ~ a finite relational signature. E.g.,
  - { PARTICIPANT(name, email, flight-nr), FLIGHT(flight-nr, dept-time) }
- Database instance (of a given schema) ~ a finite structure.
- Database queries ~ logical formulas with free variables
  - $\phi(x,y) = \exists z,u\ (\text{PARTICIPANT}(x,y,z)\ \&\ \text{FLIGHT}(z, \text{3:00AM}))$
- Database constraints ~ logical sentences expressing structural properties
  - $\forall x,y,z,u\ (\text{PARTICIPANT}(x,y,z) \rightarrow \exists t\ \text{FLIGHT}(z,t))$
  - $\forall x,y,z\ (\text{FLIGHT}(x,y)\ \&\ \text{FLIGHT}(x,z) \rightarrow y=z)$

The Data Interoperability Challenge

- Data-Interoperability:
  - Data may be distributed over different sources, using different schemas.
  - Applications need to access all these data.

The Data Interoperability Challenge

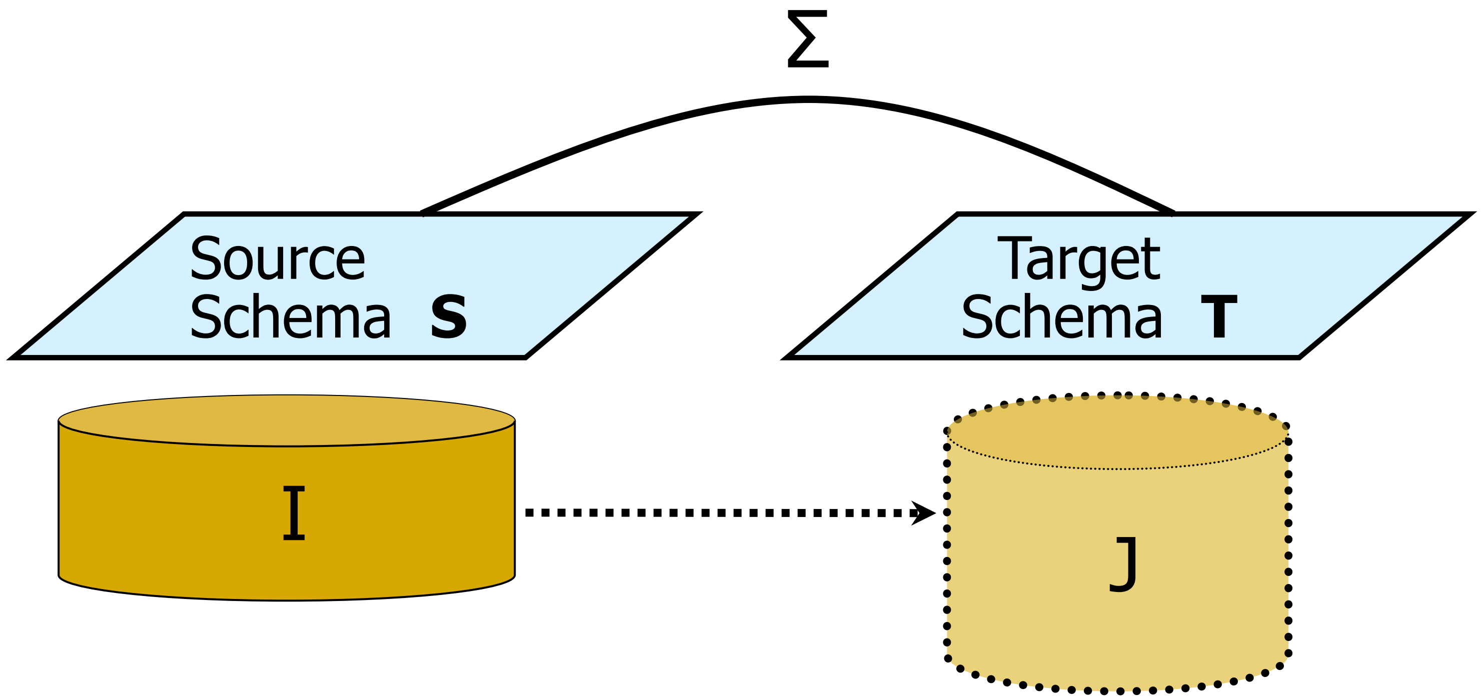
- Data-Interoperability:
  - Data may be distributed over different sources, using different schemas.
  - Applications need to access all these data.
- How can we uniformly access and manipulate data across sources?

The Data Interoperability Challenge

- Data-Interoperability:
  - Data may be distributed over different sources, using different schemas.
  - Applications need to access all these data.
- How can we uniformly access and manipulate data across sources?
- Two examples of data interoperability tasks:
  - Data Integration
  - Data Exchange

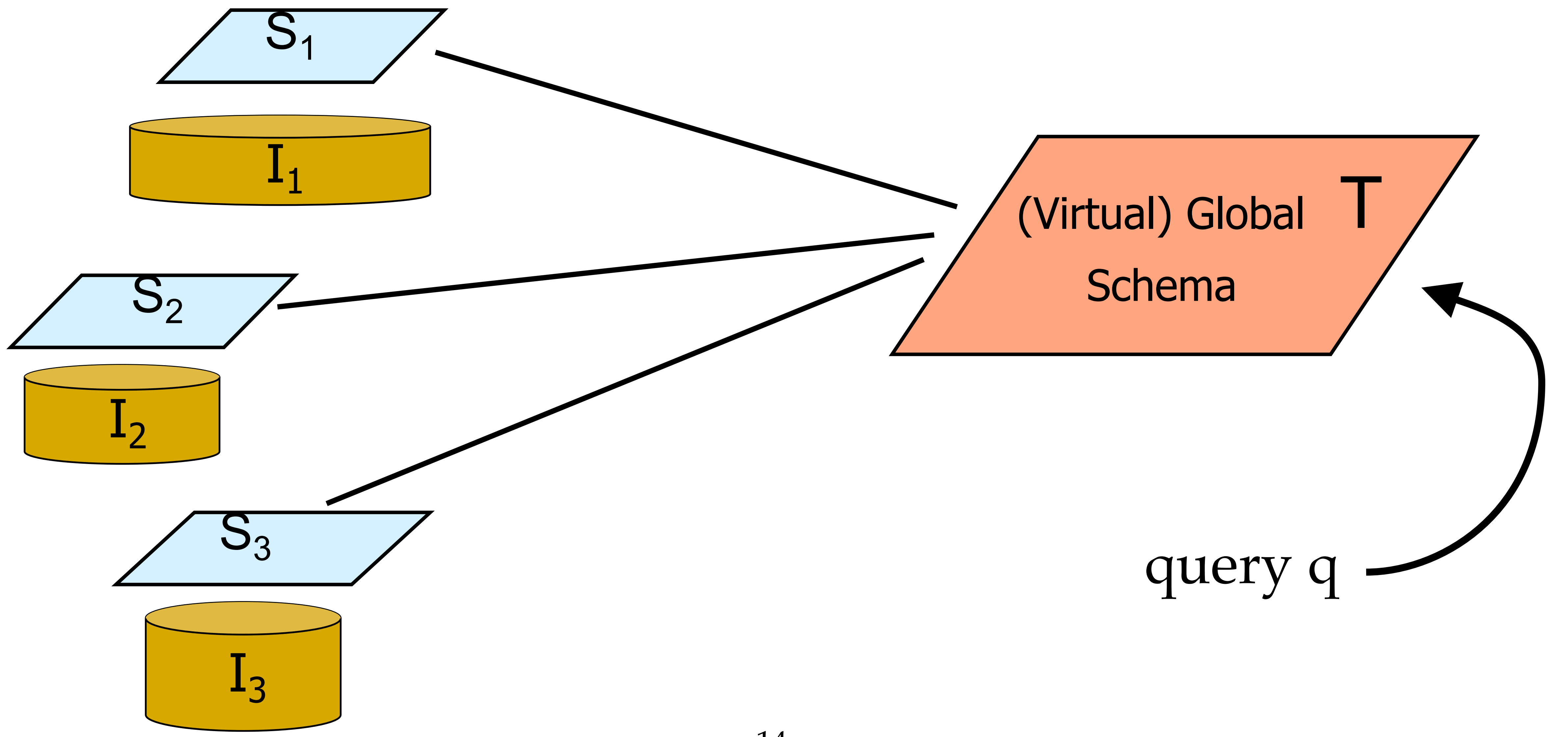
Data Exchange

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



Data Integration

Query heterogeneous data in different sources via a virtual global schema



Schema Mappings

- A schema mappings is a logical specification of the relationships between two database schemas.
- Schema mappings are fundamental in the formalization data interoperability tasks such as data exchange and data integration.

Schema Mappings

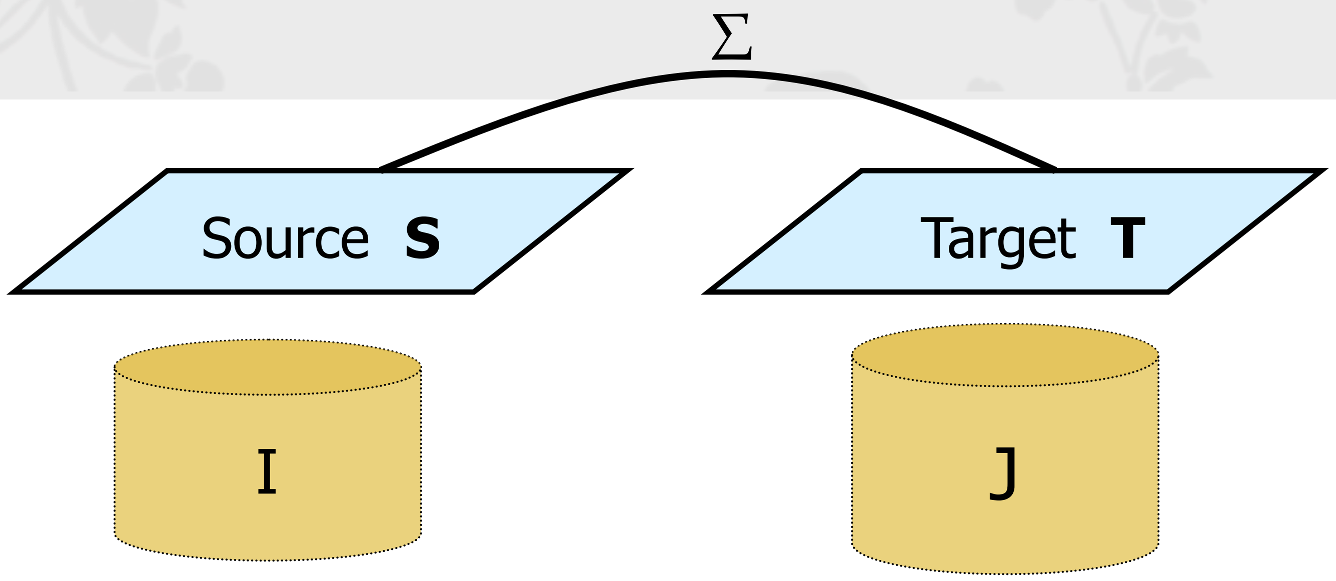
- A schema mappings is a logical specification of the relationships between two database schemas.
- Schema mappings are fundamental in the formalization data interoperability tasks such as data exchange and data integration.
- Formally, a schema mapping is a triple M=(S,T,Σ), where
  - S and T are schemas (the “source schema” and the “target schema”)
  - Σ is a collection of constraints involving the relations of S and T, specified in some schema mapping language (details to come). E.g., ∀x,y,z(PARTICIPANT(x,y,z) → MAILINGLIST(x,y)).

Schema Mapping Languages

- The choice of schema mapping language involves a compromise between expressive power and practical usability.
  - Allowing arbitrary FO sentences in Σ would make the interesting problems undecidable.

- Two of the most important schema mapping specification languages:
  - **GLAV constraints**. These are dependencies $\forall \mathbf{x}\, (\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\, \psi(\mathbf{x},\mathbf{y}))$ where
    - ϕ is a conjunction of relational atomic formulas over the source schema
    - ψ is a conjunction of relational atomic formulas over the target schema.
  - **GAV constraints**: special case of GLAV where the consequent is a single atomic formula (no existential quantification)
  - **LAV constraints:** special case of GLAV where the antecedent is a single atomic formula.

Semantics of Schema Mappings



- M = (S, T, Σ) schema mapping
  with Σ a set of GLAV constraints.

- From a semantic point of view, M can be identified with the set of all its positive data examples.

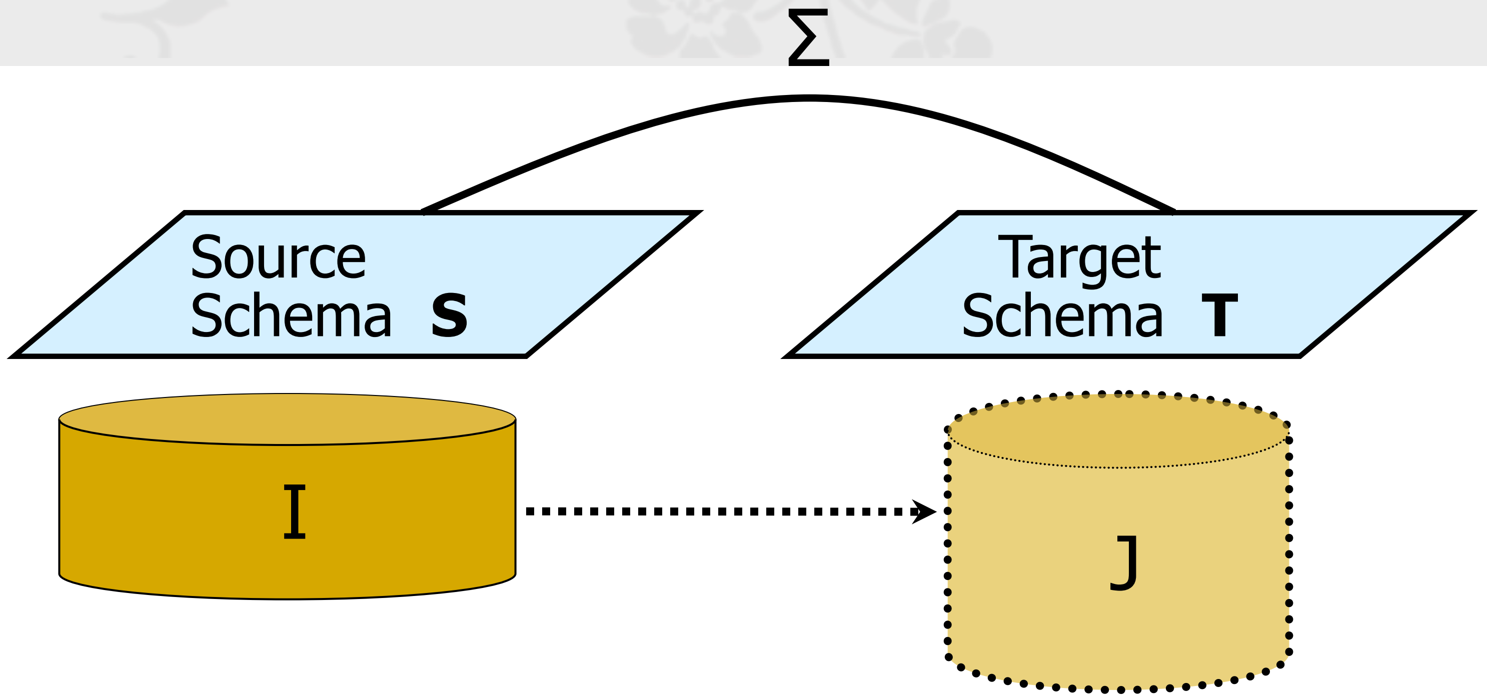
  - Data Example: A pair (I,J) where I is a source instance and J is a target instance.
  - Positive Data Example for M: a data example (I,J) such that (I,J) ⊨ Σ
  - Negative Data Example for M: a data example (I,J) such that (I,J) ⊨ Σ
  - If (I,J) is a positive data example for M, we say that J is a solution for I w.r.t. M.

Sem(M) = { (I,J): J is a solution for I w.r.t. M }

Examples

- Consider the schema mapping M = ({E}, {F}, Σ), where
  - $\Sigma = \{\ E(x,y) \rightarrow \exists z\,(F(x,z) \land F(z,y))\ \}$
- Positive Data Examples (I,J)   (i.e., J a solution for I w.r.t. M)
  - I = { E(1,2) } J = { F(1,1), F(1,2) }
  - I = { E(1,2) } J = { F(1,xxx), F(xxx,2) }
  - I = { E(1,2) } J = { F(1,xxx), F(xxx,2), F(2,3) }
- Negative Data Examples (I,J)   (i.e., J not a solution for I w.r.t. M)
  - I = { E(1,2) } J = { F(1,3) }
  - I = { E(1,2) } J = { F(1,3), F(4,2) }

Data Exchange via a Schema Mapping



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

  Given a source instance I, construct a solution J for I.

- Difficulty:
  - Typically, there are multiple solutions
  - Which one is the "best" to materialize?

Data Exchange & Universal solutions

Fagin, Kolaitis, Miller, Popa (2003):

Identified and studied the concept of a **universal solution** in data exchange.

- A universal solution is a most general solution.
- A universal solution "represents" the entire space of solutions.

Universal Solutions in Data Exchange

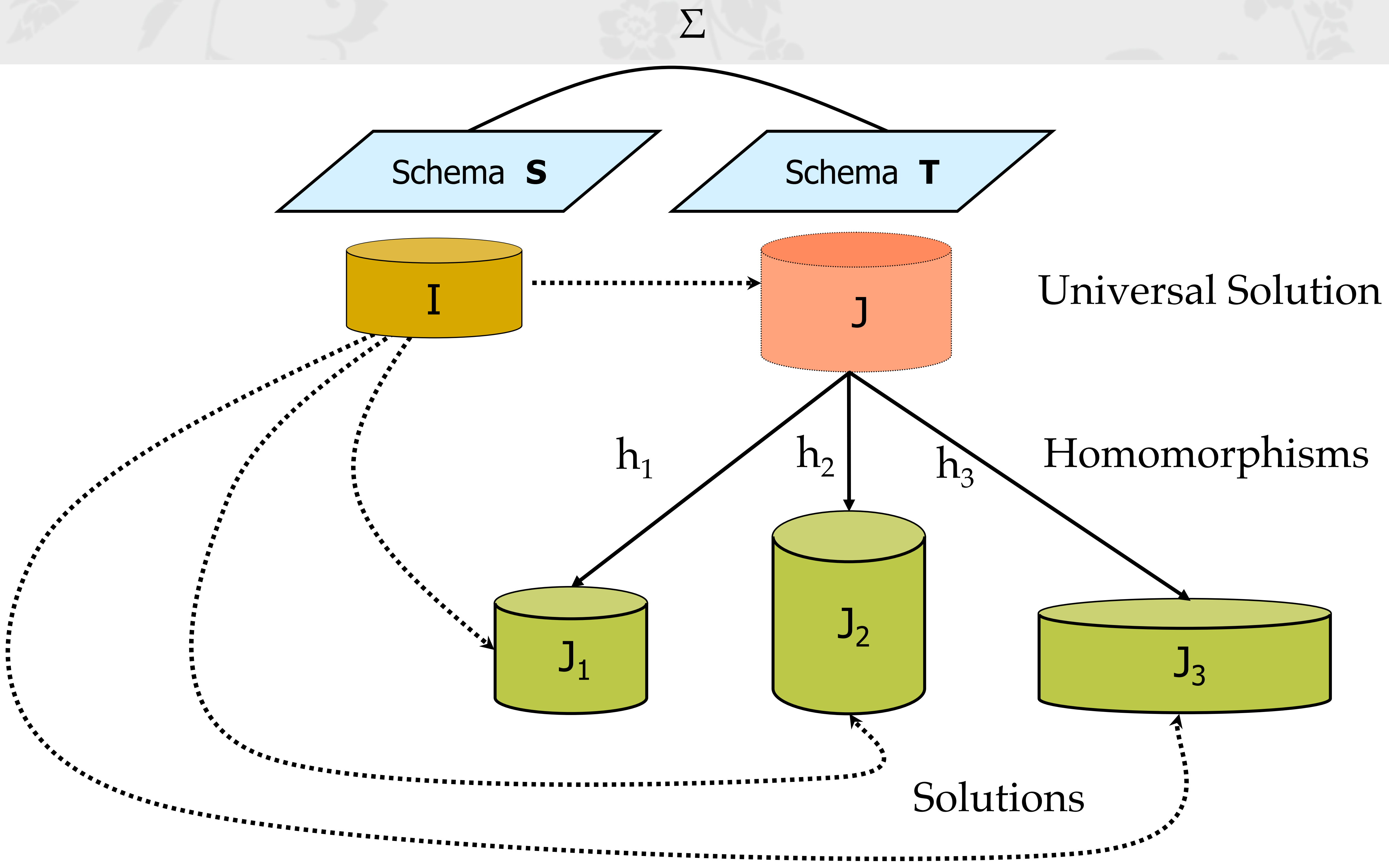
Allow two types of values in instances: **constant values** and (**labelled**) **null values.**

Definition (FKMP): A solution J for I is **universal** if it has homomorphisms to all other solutions for I, where the homomorphism may only change the null values.

(thus, a universal solution is a "most general" solution).

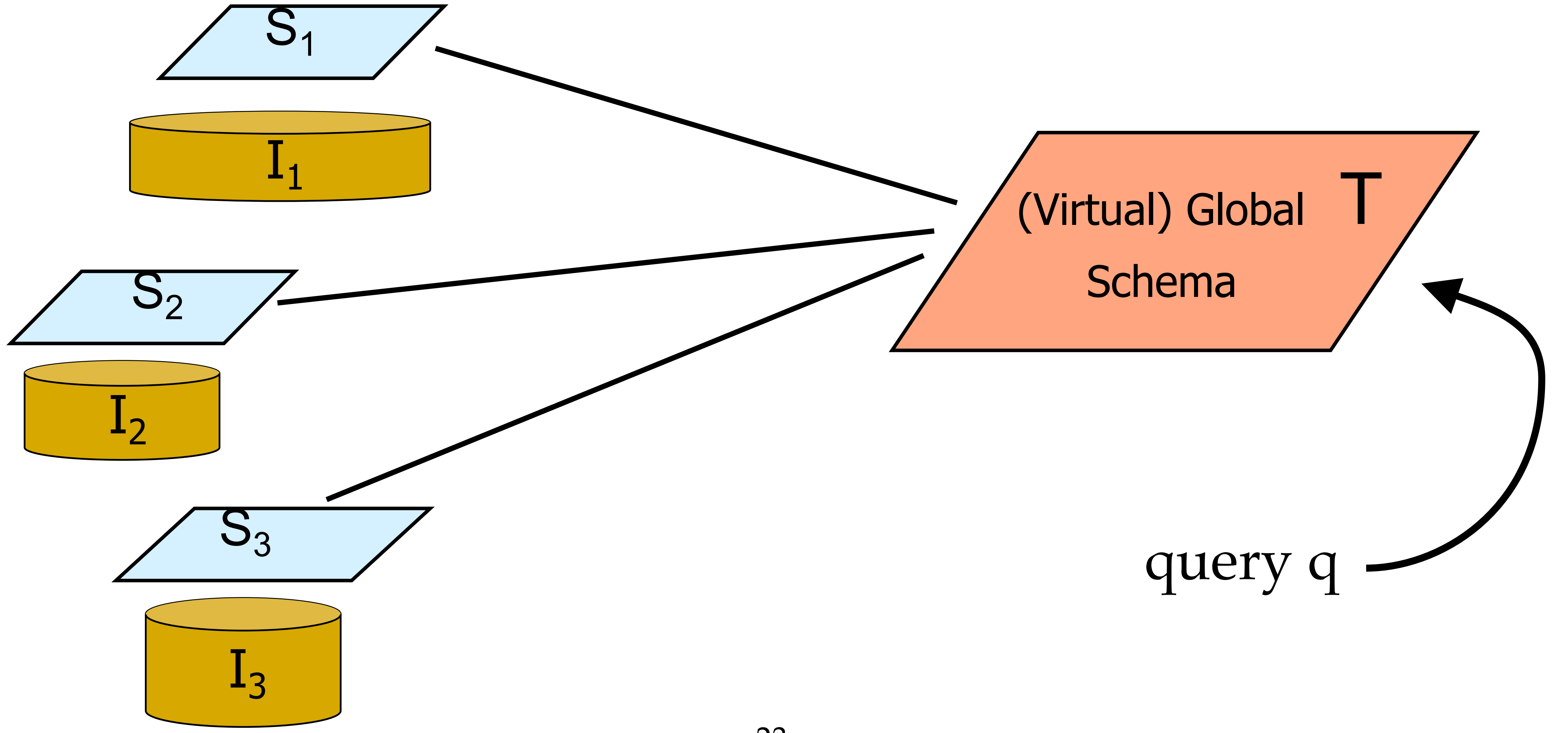
Basic result (FKMP): Universal solutions can be constructed in PTIME (data complexity) using an algorithm called the chase.

Universal Solutions in Data Exchange



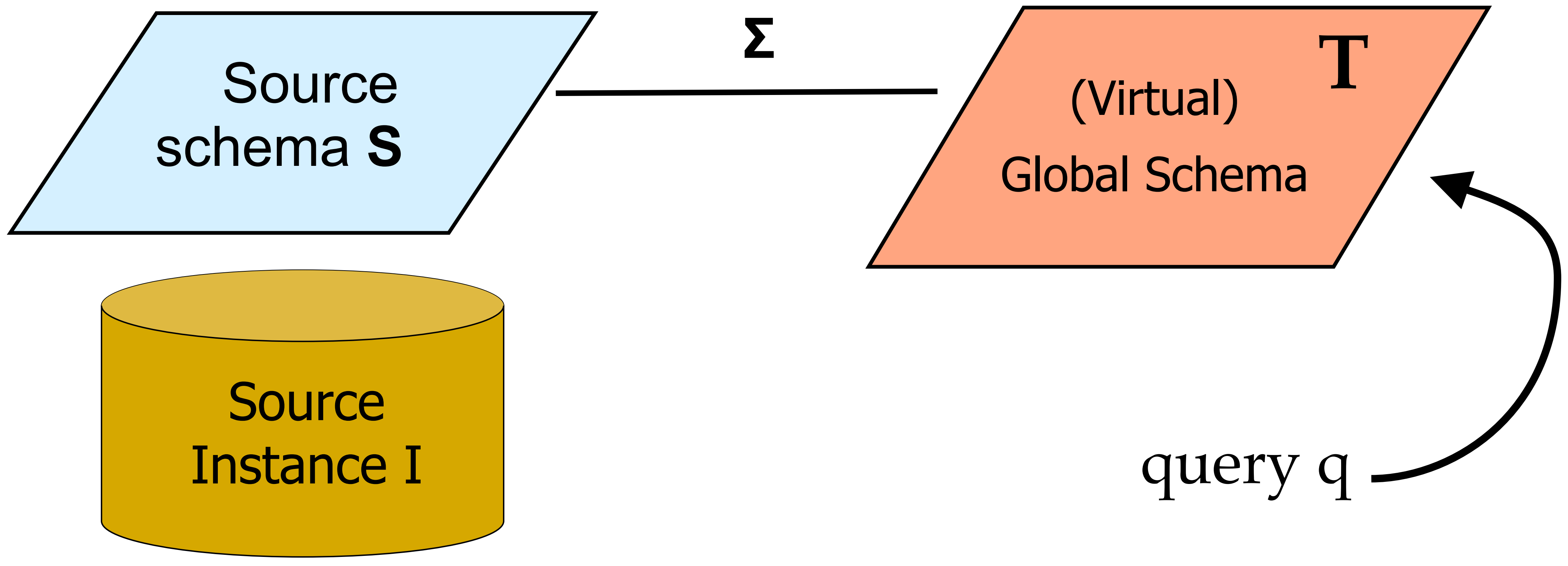
Data Integration

Query heterogeneous data in different sources via a virtual global schema



Data Integration

Query heterogeneous data in different sources via a virtual global schema



Certain answers

- Let I be a source instance and let q be a target query (a query over **T**).

- Definition: $\text{certain}_{\mathbf{M}}(q,I) = \bigcap\{q(J) \mid J \text{ solution of } I \text{ w.r.t. } \mathbf{M}\}$

  - Idea: $\text{certain}_{\mathbf{M}}(q,I)$ contains the tuples that belong to the answer of q in **all** solutions of I.

Certain answers

- Let I be a source instance and let q be a target query (a query over **T**).

- **Definition**: $\text{certain}_{\mathbf{M}}(q,I) = \bigcap\{q(J) \mid J \text{ solution of } I \text{ w.r.t. } \mathbf{M}\}$

  - Idea: $\text{certain}_{\mathbf{M}}(q,I)$ contains the tuples that belong to the answer of q in **all** solutions of I.

- If the query is a UCQ, then $\text{certain}_{\mathbf{M}}(q,I)$ can be computed in PTIME.

  - via universal solutions or via query rewriting

Computing certain answers

- **Theorem** (Fagin, Kolaitis, Miller, Popa 2003):
  - Let J be a universal solution of I w.r.t. **M**. Then for every UCQ q, $\text{certain}_{\mathbf{M}}(q,I) = q(J)_{\downarrow}$

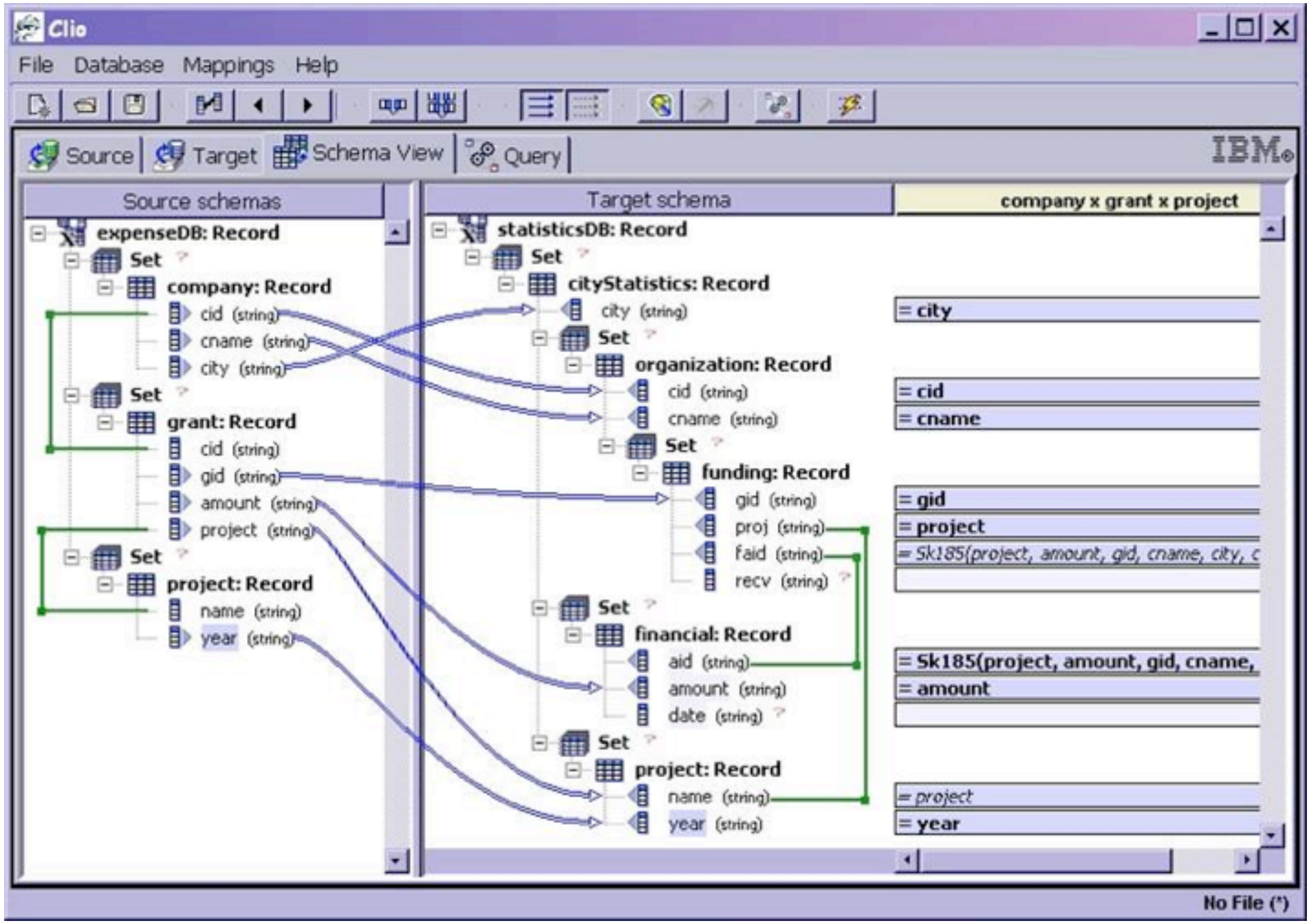
- **Theorem** (Abiteboul, Duschka 1998 ++) :
  - For every target UCQ q, there is a source UCQ q' such that $q'(I) = \text{certain}_{\mathbf{M}}(q,I)$.

Where to get your schema mapping

- Constructing a schema mapping is the first step in data exchange and data integration.

Where to get your schema mapping

- Constructing a schema mapping is the first step in data exchange and data integration.
- **Common approach** (Clio, HepToX, Microsoft mapping composer):
  - derive a schema mapping from a schema matching (a collection of correspondences between attributes of the two schemas).
  - The schema matching itself is obtained semi-automatically using schema matching techniques or by interaction with a user.
  - NB: a schema matching does not uniquely determine a schema mapping.



Source schemas

```
expenseDB: Record
  Set
    company: Record
      cid (string)
      cname (string)
      city (string)
  Set
    grant: Record
      cid (string)
      gid (string)
      amount (string)
      project (string)
  Set
    project: Record
      name (string)
      year (string)
```

Target schema | company x grant x project

```
statisticsDB: Record
  Set
    cityStatistics: Record
      city (string)                 = city
      Set
        organization: Record
          cid (string)              = cid
          cname (string)            = cname
          Set
            funding: Record
              gid (string)          = gid
              proj (string)         = project
              faid (string)         = Sk185(project, amount, gid, cname, city, c
              recv (string)
      Set
        financial: Record
          aid (string)              = Sk185(project, amount, gid, cname,
          amount (string)           = amount
          date (string)
      Set
        project: Record
          name (string)             = project
          year (string)             = year
```

Data Examples

- Using data examples in schema mapping design:
  - Data examples can be used to illustrate a candidate schema mapping
  - Deriving schema mappings from examples (learning problem)

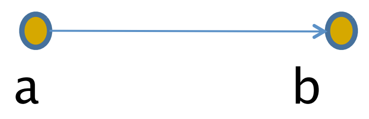
- Labeled data examples: a data example (I,J) labeled as being
  - positive -- meaning that J is a solution for I,
  - negative -- meaning that J is not a solution for I, or
  - universal -- meaning J is a universal solution for I.

Uniquely Characterizing Data Examples

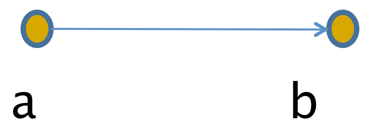
- A set E of labeled data examples uniquely characterizes a schema mapping M, within a class of schema mappings C, if
  - M fits all data examples in E.
  - every schema mapping M′ ∈ C that fits all examples in E is logically equivalent to M.

- Let M be the schema mapping specified by the GLAV constraint $\forall x,y\ (E(x,y) \rightarrow F(x,y))$.
    - This is both a GAV schema mapping and a LAV schema mapping.
  - The universal data example (I,J) with $I = \{ E(a,b) \}, J = \{ F(a,b) \}$ uniquely characterizes M w.r.t. the class of all LAV constraints.
  - There is a finite set of universal examples that uniquely characterizes M w.r.t. the class of all GAV constraints.
  - There is no finite set of universal examples that uniquely characterizes M w.r.t. the class of all GLAV constraints.

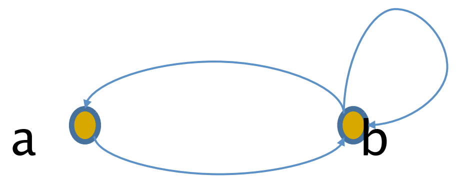
I_1



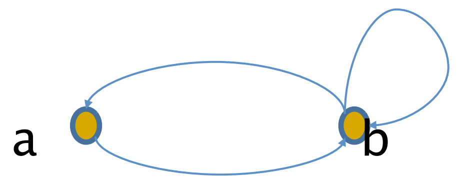
J_1



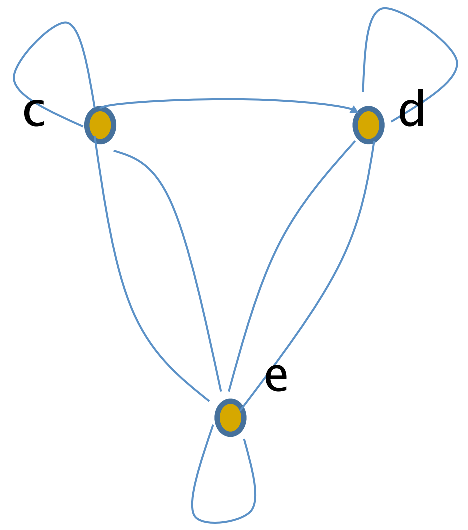
I_2



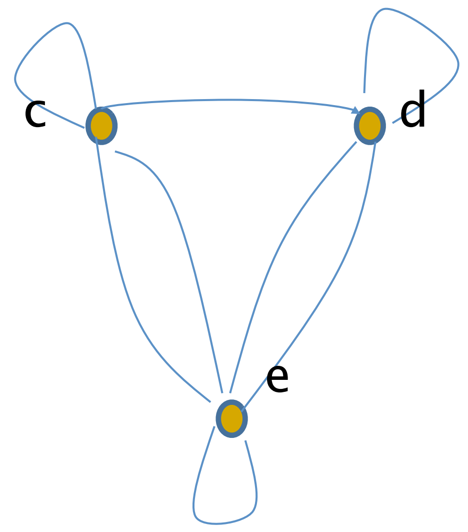
J_2



I_3



J_3



- **Problem**: which GAV schema mappings are uniquely characterizable, by a finite set of labeled data examples, within the class of GAV schema mappings?

- The solution was obtained through an intimate connection with dualities in the homomorphism lattice.

More about homomorphisms

More about homomorphisms

- Fix a schema S.
  - When we speak of structures, we will mean finite structures over S.
  - We will assume that S contains at least one non-unary relation symbol.

More about homomorphisms

- Fix a schema S.
  - When we speak of structures, we will mean finite structures over S.
  - We will assume that S contains at least one non-unary relation symbol.
- Recall: (FinStr[S], $\rightarrow$) is a quasi-order (reflexive and transitive).

More about homomorphisms

- Fix a schema S.
  - When we speak of structures, we will mean finite structures over S.
  - We will assume that S contains at least one non-unary relation symbol.
- Recall: (FinStr[S], $\rightarrow$) is a quasi-order (reflexive and transitive).
- We can construct a partially ordered set (poset) by taking the homomorphic equivalence classes.

More about homomorphisms

- Fix a schema S.
  - When we speak of structures, we will mean finite structures over S.
  - We will assume that S contains at least one non-unary relation symbol.
- Recall: (FinStr[S], $\rightarrow$) is a quasi-order (reflexive and transitive).
- We can construct a partially ordered set (poset) by taking the homomorphic equivalence classes.
- However, it turns out there is a nicer way to present this poset.

The Core of a Structure

- **Definition**:
  - The core of a (finite) structure I, denoted core(I), is the smallest substructure of I that is homomorphically equivalent to I.
  - A structure I is a core if I=core(I).

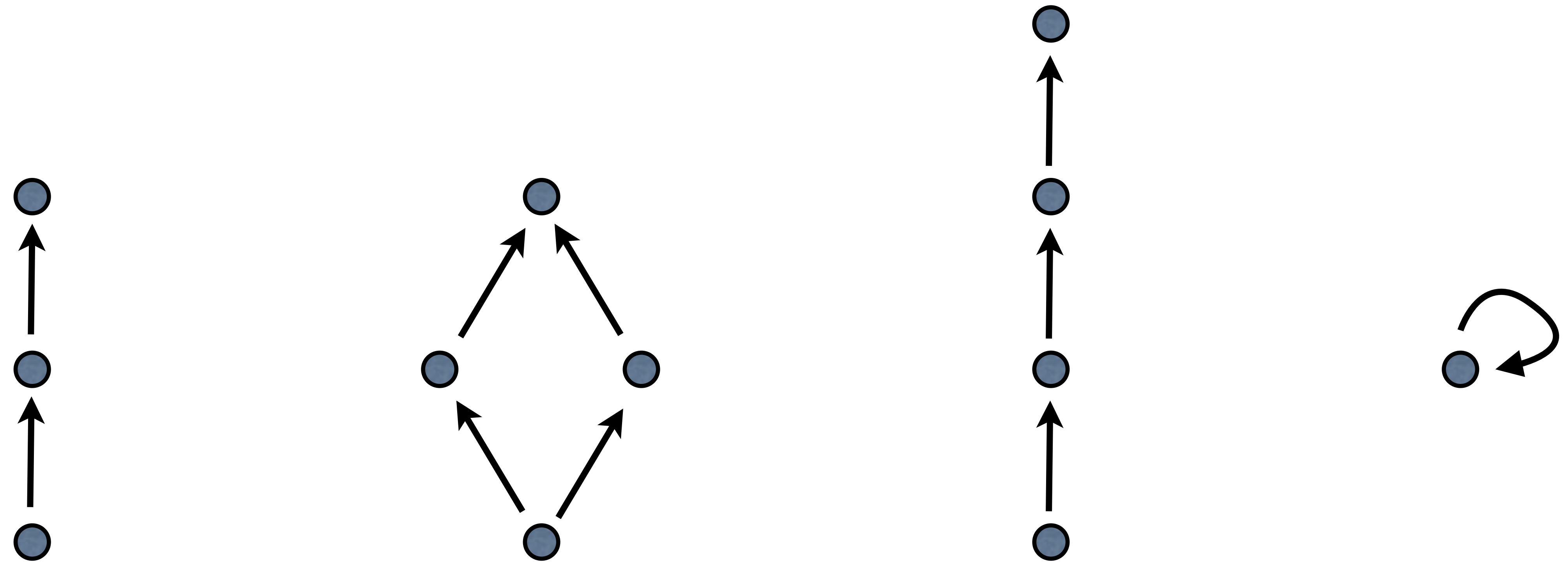
The Core of a Structure

- **Definition**:
  - The core of a (finite) structure I, denoted core(I), is the smallest substructure of I that is homomorphically equivalent to I.
  - A structure I is a core if I=core(I).
- **Theorem** [Hell and Nesetril 1992]:
  - core(I) always exists and is unique up to isomorphism
  - I $\rightleftarrows$ J iff core(I) and core(J) are isomorphic.

The Core of a Structure

- **Definition**:
  - The core of a (finite) structure I, denoted core(I), is the smallest substructure of I that is homomorphically equivalent to I.
  - A structure I is a core if I=core(I).
- **Theorem** [Hell and Nesetril 1992]:
  - core(I) always exists and is unique up to isomorphism
  - I $\rightleftarrows$ J iff core(I) and core(J) are isomorphic.
- **Corollary**:
  - if I and J are cores and I $\rightleftarrows$ J then I and J are isomorphic.
  - every ~-equivalence class has a unique (up to isomorphism) smallest representative which is a core.

Examples



The Homomorphism Lattice.

- Let CoreStr[S] be the set of all non-isomorphic (finite) core structures over schema S. Then (CoreStr[S],→) is a poset, and in fact a lattice.

The Homomorphism Lattice.

- Let CoreStr[S] be the set of all non-isomorphic (finite) core structures over schema S. Then (CoreStr[S],→) is a poset, and in fact a lattice.
- This lattice has been extensively studied. For example:
  - **Theorem** [Pultr and Trnkova 1980]: Every countable poset is isomorphic to a suborder of (CoreStr[S],→)

- $\rightarrow I \quad = \{J : J \rightarrow I\}$

- $\rightarrow I = \{J : J \rightarrow I\}$
  - Example (for graphs): $\rightarrow \mathbf{K}_2$ = Class of 2-colorable graphs

- $\rightarrow I = \{J : J \rightarrow I\}$
  - Example (for graphs): $\rightarrow \mathbf{K}_2$ = Class of 2-colorable graphs
- $I \rightarrow = \{J : I \rightarrow J\}$

- $\rightarrow I \quad = \{J : J \rightarrow I\}$
  - Example (for graphs): $\rightarrow \mathbf{K}_2$ = Class of 2-colorable graphs
- $I \rightarrow \quad = \{J : I \rightarrow J\}$
  - Example (for graphs): $\mathbf{K}_2 \rightarrow$ = Class of graphs with at least one edge.

- $\rightarrow I \quad = \{J : J \rightarrow I\}$
  - Example (for graphs): $\rightarrow \mathbf{K}_2$ = Class of 2-colorable graphs
- $I\rightarrow \quad = \{J: I \rightarrow J\}$
  - Example (for graphs): $\mathbf{K}_2\rightarrow$ = Class of graphs with at least one edge.

- $\rightarrow I = \{J : J \rightarrow I\}$
  - Example (for graphs): $\rightarrow \mathbf{K}_2$ = Class of 2-colorable graphs
- $I \rightarrow = \{J : I \rightarrow J\}$
  - Example (for graphs): $\mathbf{K}_2 \rightarrow$ = Class of graphs with at least one edge.

- Note:

- $\rightarrow I \quad = \{J : J \rightarrow I\}$
  - Example (for graphs): $\rightarrow \mathbf{K}_2$ = Class of 2-colorable graphs
- $I\rightarrow \quad = \{J : I \rightarrow J\}$
  - Example (for graphs): $\mathbf{K}_2\rightarrow$ = Class of graphs with at least one edge.

- Note:
  - $\rightarrow I$ defines a downward closed set in the homomorphism lattice.

- $\to I \quad = \{J : J \to I\}$
  - Example (for graphs): $\to \mathbf{K}_2$ = Class of 2-colorable graphs
- $I \to \quad = \{J : I \to J\}$
  - Example (for graphs): $\mathbf{K}_2 \to$ = Class of graphs with at least one edge.

- Note:
  - $\to I$ defines a downward closed set in the homomorphism lattice.
  - $I \to$ defines an upward closed set in the homomorphism lattice.

- $\rightarrow I \quad = \{J : J \rightarrow I\}$
  - Example (for graphs): $\rightarrow \mathbf{K}_2 \; = \;$ Class of 2-colorable graphs
- $I \rightarrow \quad = \{J : I \rightarrow J\}$
  - Example (for graphs): $\mathbf{K}_2 \rightarrow \; = \;$ Class of graphs with at least one edge.

- Note:
  - $\rightarrow I$ defines a downward closed set in the homomorphism lattice.
  - $I \rightarrow$ defines an upward closed set in the homomorphism lattice.
  - $I \nrightarrow$ defines a downward closed set in the homomorphism lattice.

Simple Duality Pairs

- **Definition**: Let D and F be two finite structures
  - (F,D) is a duality pair if $\rightarrow D = F \not\rightarrow$
  - In other words, for every structure I, I $\rightarrow$ D if and only if F $\not\rightarrow$ I.
  - In this case, we say that F is an obstruction for D.

Simple Duality Pairs

- **Definition**: Let D and F be two finite structures
  - (F,D) is a duality pair if $\rightarrow D = F \not\rightarrow$
  - In other words, for every structure I, $I \rightarrow D$ if and only if $F \not\rightarrow I$.
  - In this case, we say that F is an obstruction for D.

- **Example**:
  - For graphs, $(\mathbf{K}_2, \mathbf{K}_1)$ is a duality pair

- **Gallai-Hasse-Roy-Vitaver Theorem** (~1965) for directed graphs:
  - Let $\mathbf{T}_k$ be the linear order with k elements, $\mathbf{P}_{k+1}$ be the path with k+1 elements. Then ($\mathbf{P}_{k+1}$, $\mathbf{T}_k$) is a duality pair, since for every directed graphs H, H $\rightarrow$ $\mathbf{T}_k$ if and only if $\mathbf{P}_{k+1} \nrightarrow$ H.

Duality Pairs

- **Theorem** (König 1936): A graph is 2-colorable if and only if it contains no cycle of odd length. In symbols, $\rightarrow \mathbf{K}_2 = \bigcap_{i\geq 0} (\mathbf{C}_{2i+1} \nrightarrow)$.

Duality Pairs

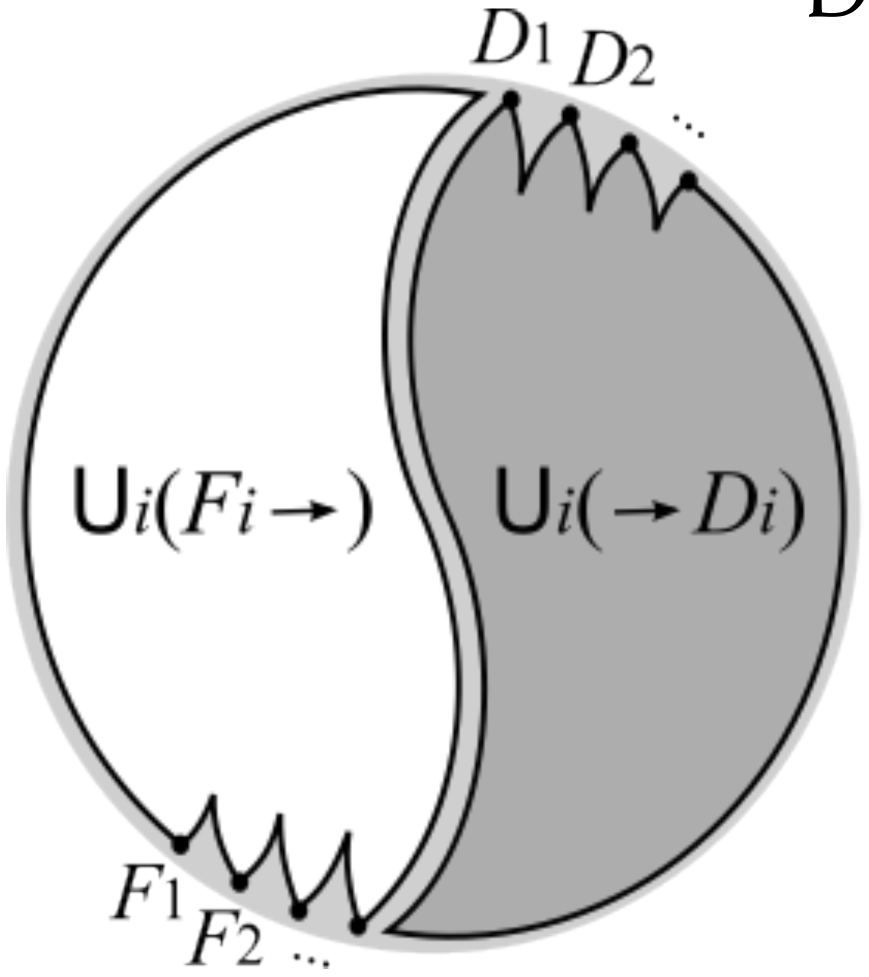
- **Theorem** (König 1936): A graph is 2-colorable if and only if it contains no cycle of odd length. In symbols, $\rightarrow \mathbf{K}_2 = \bigcap_{i\geq 0} (\mathbf{C}_{2i+1} \nrightarrow)$.
- **Definition**: Let *F* and *D* be two sets of structures. We say that (***F***, ***D***) is a duality pair if $\bigcup_{D \in \boldsymbol{D}} (\rightarrow D) = \bigcap_{F \in \boldsymbol{F}} (F \nrightarrow)$.
  - In other words, for every structure I, tfae:
    - There is a structure D in ***D*** such that I $\rightarrow$ D.
    - For every structure F in ***F***, we have F $\nrightarrow$ I.
  - In this case, we say that F is an obstruction set for D.

Duality Pair ($\boldsymbol{F}$,$\boldsymbol{D}$),where

$\boldsymbol{F} = \{F_1, F_2, \ldots\}$

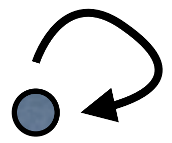
$\boldsymbol{D} = \{D_1, D_2, \ldots\}$

“Desires”



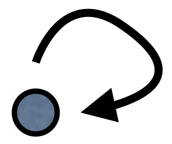
“Frustrations”

Example



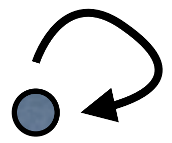
- Let F be the one-element cycle.

Example



- Let F be the one-element cycle.
- **Question**: Is {F} an obstruction set for a finite set of structures?
  - I.e., is there a duality pair of the form ({F},D) ?

Example



- Let F be the one-element cycle.
- **Question**: Is {F} an obstruction set for a finite set of structures?
  - I.e., is there a duality pair of the form ({F},D) ?
- No. This has to do with the fact that F contains a cycle.

Acyclicity

- The incidence graph inc(A) of a structure A is the bipartite graph with
  - nodes: the elements of A and the atomic facts (e.g., $R(a_1,\ldots,a_n)$) of A
  - edges between elements and facts in which they occur

Acyclicity

- The incidence graph inc(A) of a structure A is the bipartite graph with
  - nodes: the elements of A and the atomic facts (e.g., $R(a_1,\ldots,a_n)$) of A
  - edges between elements and facts in which they occur
- The structure A is acyclic if
  - Inc(A) is acyclic, and
  - No element occurs twice in the the same fact.

Characterization of Obstruction Sets

- **Theorem** (Foniok, Nešetřil, and Tardif 2008):
  - Let ***F*** be a finite set of homomorphically incomparable core structures. Tfae:
    - ***F*** is an obstruction set of some finite set ***D*** of structures.
    - Each structure in ***F*** is acyclic.
  - Moreover, there is an algorithm that, given such a set ***F*** consisting of acyclic structures, computes a finite set ***D*** of structures such that (***F***, ***D***) is a duality pair.

Characterization of Obstruction Sets

- **Theorem** (Foniok, Nešetřil, and Tardif 2008):
  - Let ***F*** be a finite set of homomorphically incomparable core structures. Tfae:
    - ***F*** is an obstruction set of some finite set ***D*** of structures.
    - Each structure in ***F*** is acyclic.
  - Moreover, there is an algorithm that, given such a set ***F*** consisting of acyclic structures, computes a finite set ***D*** of structures such that (***F***, ***D***) is a duality pair.
- In particular, if F is the one-element cycle, then {F} is not an obstruction set of any finite set of structures.

Structures with Constant Symbols

Structures with Constant Symbols

- The preceding theorem extends to structures with constant symbols when acyclicity is replaced by c-acyclicity.

Structures with Constant Symbols

- The preceding theorem extends to structures with constant symbols when acyclicity is replaced by c-acyclicity.
- A structure with constant symbols is c-acyclic if
  - Every cycle in Inc(A) contains an element named by a constant symbol, and
  - Only elements named by constant symbols may occur twice in the same fact.

Back to Schema Mappings

- The canonical structure of a GAV constraint

$$\forall \mathbf{x}\, (\varphi_1(\mathbf{x}) \wedge \ldots \wedge \varphi_\kappa(\mathbf{x}) \rightarrow R(x_{i1},\ldots,x_{im}))$$

  is the structure with

  - domain: the variables in **x** themselves
  - atomic facts: $\varphi 1(x)$, ..., $\varphi\kappa(x)$
  - constant symbols $c_1,\ldots,c_m$ denoting $x_{i1},\ldots,x_{im}$

Back to Schema Mappings

- The canonical structure of a GAV constraint

  $$\forall \mathbf{x}\ (\varphi_1(\mathbf{x}) \wedge \dots \wedge \varphi_\kappa(\mathbf{x}) \rightarrow R(x_{i1},\dots,x_{im}))$$

  is the structure with

  - domain: the variables in **x** themselves
  - atomic facts: $\varphi 1(x)$, ..., $\varphi\kappa(x)$
  - constant symbols $c_1,\dots,c_m$ denoting $x_{i1},\dots,x_{im}$

- **Example**: $\forall xyz\ (E(x,y) \wedge E(y,z) \rightarrow R(x,z))$ has canonical structure

Back to Schema Mappings

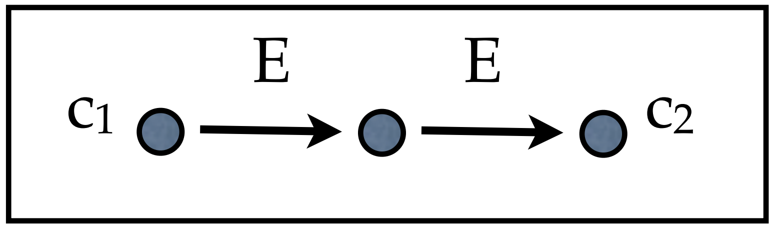
- The canonical structure of a GAV constraint

  $$\forall \mathbf{x}\ (\varphi_1(\mathbf{x}) \wedge \ldots \wedge \varphi_\kappa(\mathbf{x}) \rightarrow R(x_{i1},\ldots,x_{im}))$$

  is the structure with

  - domain: the variables in **x** themselves
  - atomic facts: $\varphi1(x)$, ..., $\varphi\kappa(x)$
  - constant symbols $c_1,\ldots,c_m$ denoting $x_{i1},\ldots,x_{im}$

- **Example**: $\forall xyz\ (E(x,y) \wedge E(y,z) \rightarrow R(x,z))$ has canonical structure



- **Theorem**: Let M = (S, T, Σ) be a GAV schema mapping. Tfae:
  - M is uniquely characterizable within the class of all GAV constraints.
  - For every target relation symbol R, the set of the canonical structures of the GAV constraints in Σ with R as their consequent is the obstruction set of some finite set D of structures.

- **Theorem**: Let M = (S, T, Σ) be a GAV schema mapping. Tfae:
  - M is uniquely characterizable within the class of all GAV constraints.
  - For every target relation symbol R, the set of the canonical structures of the GAV constraints in Σ with R as their consequent is the obstruction set of some finite set D of structures.
- **Corollary**: testing unique characterizability is NP-complete, and one can effectively construct a uniquely characterizing finite set of data examples if it exists.

Summary

- Schema mappings: a fundamental building block in the study of data-interoperability problems.
- Homomorphism dualities: a powerful tool from graph theory (with many applications in constraint satisfaction as well)

Main References

- Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, Lucian Popa (2003) Data Exchange: Semantics and Query Answering. ICDT 2003: 207-224

- Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang Chiew Tan (2011). Characterizing schema mappings via data examples. ACM Trans. Database Syst., 36(4):23

- Pavol Hell and Jaroslav Nesetril (2004). Graphs and homomorphisms.