Data exchange and schema mappings

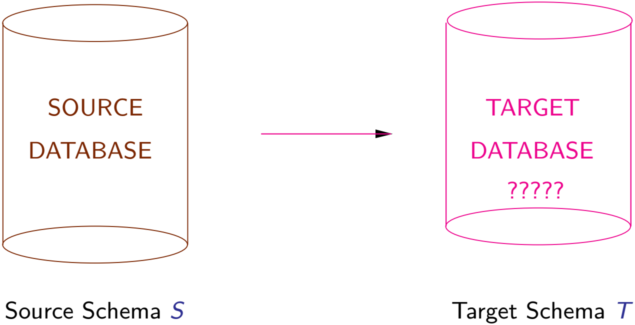
S. Amano M. Arenas L. Libkin F. Murlak

Fagin-Kolaitis-Popa-Miller, ICDT'03
Fagin-Kolaitis-Popa-Tan, PODS'04

Data exchange

- Source schema, target schema; need to transfer data between them.
- A typical scenario:
  - Two organizations have their legacy databases, schemas cannot be changed.
  - Data from organization 1 needs to be transfered to data from organization 2.
  - Queries need to be answered against the transferred data.

Data Exchange



Outline

- data exchange problem
- universal solutions
- target constraints
- composing mappings

Data exchange problem

An example

- We want to create a target database with the schema

  Flight(city1,city2,aircraft,departure,arrival)
  Served(city,country,population,agency)

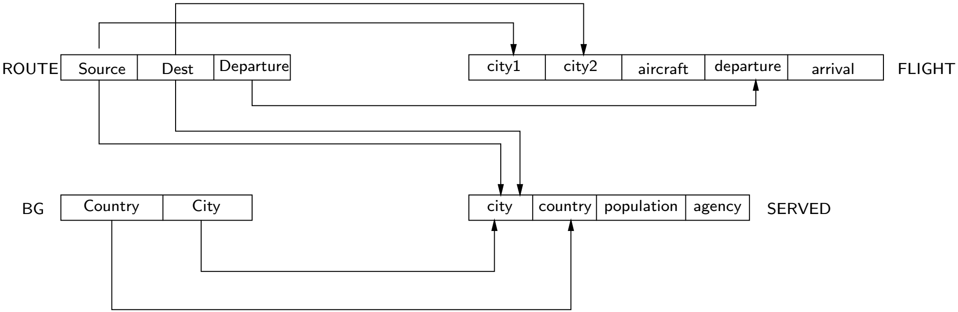
- We don't start from scratch: there is a source database containing relations

  Route(source,destination,departure)
  BG(country,city)

- We want to transfer data from the source to the target.

Relationships between source and target

How to specify the relationship?



Relationships between source and target cont'd

- Formal specification: we have a *relational calculus query* over both the source and the target schema.
- The query is of a restricted form, and can be thought of as a sequence of rules:

$$
\begin{aligned}
Route(c1, c2, dept) &\rightarrow Flight(c1, c2, \_, dept, \_) \\
Route(city, \_, \_),\ BG(city, country) &\rightarrow Served(city, country, \_, \_) \\
Route(\_, city, \_),\ BG(city, country) &\rightarrow Served(city, country, \_, \_)
\end{aligned}
$$

Targets

- Target instances should satisfy the rules.
- What does it mean to satisfy a rule?
- Formally, a source S and a target T satisfy a rule

  Route(c1, c2, dept) $\rightarrow$ *Flight(c1, c2, _ , dept, _)*

  if they satisfy the constraint

  $$\forall c_1, c_2, d \Big(Route(c_1, c_2, d) \rightarrow \exists a_1, a_2 \ (Flight(c_1, c_2, a_1, d, a_2)) \Big)$$

Targets

- What happens if there no values for some attributes, e.g. *aircraft*, *arrival*?
- We put in null values or some real values.
- But then we may have multiple solutions!

Targets

Source Database:

ROUTE:

Source	Destination	Departure
Edinburgh	Amsterdam	0600
Edinburgh	London	0615
Edinburgh	Frankfurt	0700

BG:

Country	City
UK	London
UK	Edinburgh
NL	Amsterdam
GER	Frankfurt

Look at the rule

$$\text{Route(c1, c2, dept)} \rightarrow \textit{Flight(c1, c2, \_ , dept, \_)}$$

The left hand side is satisfied by

Route(Edinburgh, Amsterdam, 0600)

But what can we put in the target?

Targets

Rule: Route(c1, c2, dept) $\rightarrow$ *Flight(c1, c2, _ , dept, _)*
Satisfied by: Route(Edinburgh, Amsterdam, 0600)
Possible targets:

- Flight(Edinburgh, Amsterdam, $\bot_1$, 0600, $\bot_2$)
- Flight(Edinburgh, Amsterdam, B737, 0600, $\bot$)
- Flight(Edinburgh, Amsterdam, $\bot$, 0600, 0845)
- Flight(Edinburgh, Amsterdam, $\bot$, 0600, $\bot$)
- Flight(Edinburgh, Amsterdam, B737, 0600, 0845)

They all satisfy the constraints!

Which target to choose?

- One of them happens to be right:
  - Flight(Edinburgh, Amsterdam, B737, 0600, 0845)
- But in general we do not know this; it looks just as good as
  - Flight(Edinburgh, Amsterdam, 'The Spirit of St Louis', 0600, 1300), or
  - Flight(Edinburgh, Amsterdam, F16, 0600, 0620).
- Goal: look for the "most general" solution.
- How to define "most general": can be mapped into any other solution.
- It is not unique either, but the space of solution is greatly reduced.
- In our case Flight(Edinburgh, Amsterdam, $\perp_1$, 0600, $\perp_2$) is most general as it makes no additional assumptions about the nulls.

Universal solutions

Universal solutions

- A homomorphism is a mapping $h : \text{Nulls} \rightarrow \text{Nulls} \cup \text{Constants}$.
- For example, $h(\perp_1) = B737$, $h(\perp_2) = 0845$.
- If we have two solutions T_1 and T_2, then h is a homomorphism from T_1 into T_2 if for each tuple t in T_1, the tuple $h(t)$ is in T_2.
- For example, if we have a tuple

$$t = \text{Flight}(\text{Edinburgh}, \text{Amsterdam}, \perp_1, 0600, \perp_2)$$

then

$$h(t) = \text{Flight}(\text{Edinburgh, Amsterdam, B737, 0600, 0845}).$$

- A solution is universal if there is a homomorphism from it into every other solution.
- (We shall revisit this definition later, to deal with nulls properly.)

Universal solutions: still too many of them

- Take any $n > 0$ and consider the solution with n tuples:

  Flight(Edinburgh, Amsterdam, $\bot_1$, 0600, $\bot_2$)
  Flight(Edinburgh, Amsterdam, $\bot_3$, 0600, $\bot_4$)
  ...
  Flight(Edinburgh, Amsterdam, $\bot_{2n-1}$, 0600, $\bot_{2n}$)

- It is universal too: take a homomorphism

$$h'(\bot_i) = \begin{cases} \bot_1 & \text{if } i \text{ is odd} \\ \bot_2 & \text{if } i \text{ is even} \end{cases}$$

- It sends this solution into

  Flight(Edinburgh, Amsterdam, $\bot_1$, 0600, $\bot_2$)

Universal solutions: cannot be distinguished by CQs

- There are queries that distinguish large and small universal solutions (e.g., does a relation have at least 2 tuples?)
- But these cannot be distinguished by conjunctive queries
- Because: if $\bot_{i_1}, \ldots, \bot_{i_k}$ witness a conjunctive query, so do $h(\bot_{i_1}), \ldots, h(\bot_{i_k})$ — hence, one tuple suffices
- In general, if we have
  - a homomorphism $h : T \to T'$,
  - a conjunctive query Q
  - a tuple t without nulls such that $t \in Q(T)$
- then $t \in Q(T')$

Universal solutions and conjunctive queries

- If
  - T and T' are two universal solutions
  - Q is a conjunctive query, and
  - t is a tuple without nulls,

  then

  $$t \in Q(T) \quad \Leftrightarrow \quad t \in Q(T')$$

  because we have homomorphisms $T \to T'$ and $T' \to T$.
- Furthermore, if
  - T is a universal solution, and T'' is an arbitrary solution,

  then

  $$t \in Q(T) \quad \Rightarrow \quad t \in Q(T'')$$

Universal solutions and conjunctive queries cont'd

- Now recall what we learned about answering conjunctive queries over databases with nulls:
  - T is a naive table
  - the set of tuples without nulls in $Q(T)$ is precisely $\text{certain}(Q, T)$ – certain answers over T
- Hence if T is an arbitrary universal solution

$$\text{certain}(Q, T) = \bigcap \{Q(T') \mid T' \text{is a solution}\}$$

- $\bigcap \{Q(T') \mid T' \text{is a solution}\}$ is the set of certain answers in data exchange under mapping M: $\text{certain}_M(Q, S)$. Thus

$$\text{certain}_M(Q, S) = \text{certain}(Q, T)$$

for every universal solution T for S under M.

Universal solutions cont'd

- To answer conjunctive queries, one needs an arbitrary universal solution.
- We saw some; intuitively, it is better to have:

  $$\text{Flight}(\text{Edinburgh}, \text{Amsterdam}, \quad \perp_1, \quad 0600, \quad \perp_2)$$

  than

  $$\begin{array}{lllll}
\text{Flight}(\text{Edinburgh}, & \text{Amsterdam}, & \perp_1, & 0600, & \perp_2) \\
\text{Flight}(\text{Edinburgh}, & \text{Amsterdam}, & \perp_3, & 0600, & \perp_4) \\
\ldots & & & & \\
\text{Flight}(\text{Edinburgh}, & \text{Amsterdam}, & \perp_{2n-1}, & 0600, & \perp_{2n})
\end{array}$$

- We now define a canonical universal solution.

Canonical universal solution

- Convert each rule into a rule of the form:

  $$\varphi(x_1, \ldots, x_k,\ y_1, \ldots, y_m) \quad \rightarrow \quad \psi(x_1, \ldots, x_k,\ z_1, \ldots, z_n)$$

  For example,

  Route(c1, c2, dept) $\rightarrow$ *Flight(c1, c2, _ , dept, _)*

  becomes

  $$Route(x_1, x_2, x_3) \quad \rightarrow \quad Flight(x_1, x_2, z_1, x_3, z_2)$$

- Evaluate $\varphi(x_1, \ldots, x_n,\ y_1, \ldots, y_m)$ in S.
- For each tuple $(a_1, \ldots, a_n,\ b_1, \ldots, b_m)$ that belongs to the result (i.e. $\varphi(a_1, \ldots, a_n,\ b_1, \ldots, b_m)$ holds in S), do the following:

Canonical universal solution cont'd

- ... do the following:
  - Create new (not previously used) null values $\bot_1, \ldots, \bot_k$
  - Put tuples in target relations so that

    $$\psi(a_1, \ldots, a_n,\ \bot_1, \ldots, \bot_k)$$

    holds.
- What is ψ?
- It is normally assumed that ψ is a conjunction of atomic formulae, i.e.

  $$R_1(\bar{x}_1, \bar{z}_1) \wedge \ldots \wedge R_l(\bar{x}_l, \bar{z}_l)$$

- Tuples are put in the target to satisfy these formulae

Canonical universal solution cont'd

- Example: no-direct-route airline:

$$\text{Oldroute}(x_1, x_2) \;\rightarrow\; \text{Newroute}(x_1, z) \wedge \text{Newroute}(z, x_2)$$

- If $(a_1, a_2) \in \text{Oldroute}(a_1, a_2)$, create a new null $\bot$ and put

  $\text{Newroute}(a_1, \bot)$
  $\text{Newroute}(\bot, a_2)$

  into the target.

- Complexity of finding this solution: polynomial in the size of the source S:

$$O\left(\sum_{\text{rules } \varphi \,\rightarrow\, \psi} \text{Evaluation of } \varphi \text{ on } S \right)$$

Canonical universal solution and conjunctive queries

- Canonical solution: $\textsc{CanSol}_M(S)$.
- We know that if Q is a conjunctive query, then $\text{certain}_M(Q, S) \;=\; \text{certain}(Q, T)$ for every universal solution T for S under M.
- Hence

$$\text{certain}_M(Q, S) \quad = \quad \text{certain}(Q, \textsc{CanSol}_M(S))$$

- Algorithm for answering Q:
  - Construct $\textsc{CanSol}_M(S)$
  - Apply naive evaluation to Q over $\textsc{CanSol}_M(S)$

Target constraints

Data exchange and integrity constraints

- Integrity constraints are often specified over target schemas
- In SQL's data definition language one uses keys and foreign keys most often, but other constraints can be specified too.
- Adding integrity constraints in data exchange is often problematic, as some natural solutions – e.g., the canonical solution – may fail them.

Target constraints cause problems

- The simplest example:
  - Copy source to target
  - Impose a constraint on target not satisfied in the source
- Schema mapping:
  - $S(x, y) \rightarrow T(x, y)$ and
  - Constraint: the first attribute is a key
- Instance S:

1	2
1	3

- Every target T must include these tuples and hence violates the key.

Target constraints: more problems

A common problem: an attempt to repair violations of constraints leads to a sequence of tuple insertions.

- Source DeptEmpl(dept_id,manager_name,empl_id)
- Target
  - Dept(dept_id,manager_id,manager_name),
  - Empl(empl_id,dept_id)
- Rule $\text{DeptEmpl}(d, n, e) \quad \rightarrow \quad \text{Dept}(d, z, n) \wedge \text{Empl}(e, d)$
- Target constraints:
  - Dept[manager_id] $\subseteq$ Empl[empl_id]
  - Empl[dept_id] $\subseteq$ Dept[dept_id]

Target constraints: more problems cont'd

- Start with (CS, John, 001) in DeptEmpl.
- Put Dept(CS, $\perp_1$, John) and Empl(001, CS) in the target
- Use the first constraint and add a tuple Empl($\perp_1$, $\perp_2$) in the target
- Use the second constraint and put Dept($\perp_2$, $\perp_3$, $\perp_3$') into the target
- Use the first constraint and add a tuple Empl($\perp_3$, $\perp_4$) in the target
- Use the second constraint and put Dept($\perp_4$, $\perp_5$, $\perp_5$') into the target
- this never stops....

Target constraints: avoiding this problem

- Change the target constraints slightly:
  - Target constraints:
    - Dept[dept_id,manager_id] ⊆ Empl[empl_id, dept_id]
    - Empl[dept_id] ⊆ Dept[dept_id]
- Again start with (CS, John, 001) in DeptEmpl.
- Put Dept(CS, $\perp_1$, John) and Empl(001, CS) in the target
- Use the first constraint and add a tuple Empl($\perp_1$, CS)
- Now constraints are satisfied – we have a target instance!
- What's the difference? In our first example constraints are very cyclic causing an infinite loop. There is less cyclicity in the second example.

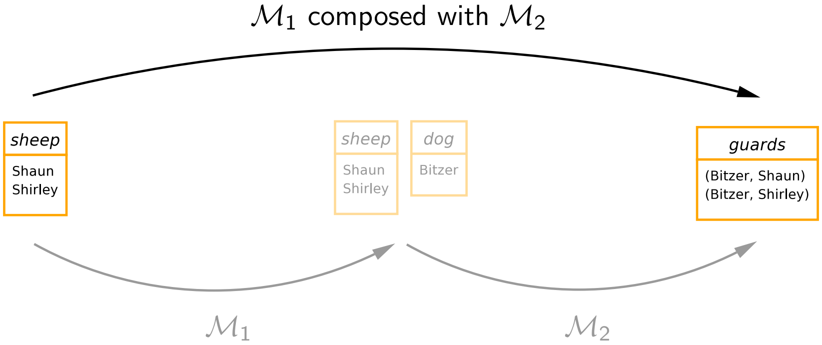
Bottom line: avoid cyclic constraints.

Composing mappings

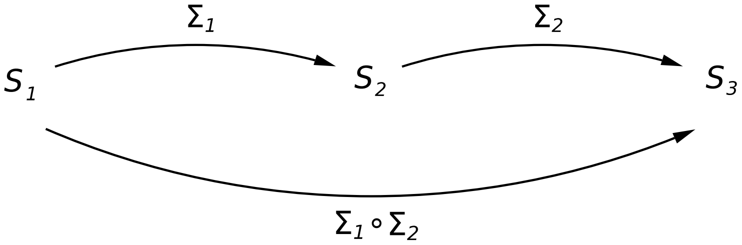
Schema mappings

- Rules used in data exchange specify mappings between schemas.
- To understand the evolution of data one needs to study operations on schema mappings.
- Most commonly we need to deal with composition.

Semantics


$\mathcal{M}_1$ composed with $\mathcal{M}_2$
sheep
Shaun
Shirley
sheep
Shaun
Shirley
dog
Bitzer
guards
(Bitzer, Shaun)
(Bitzer, Shirley)
$\mathcal{M}_1$
$\mathcal{M}_2$

The closure problem



- Are mappings closed under composition?
- If not, what needs to be added?

Composition: when it works

- Example:

$$\begin{array}{lrcl} \Sigma: & S(x_1,x_2,x_3) & \rightarrow & T(x_1,x_2) \wedge T(x_2,x_3) \\ \Delta: & T(x_1,x_2) & \rightarrow & W(x_1,x_2,z) \end{array}$$

- First modify into:

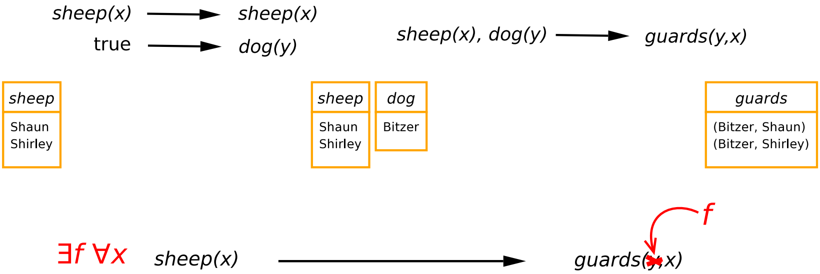
$$\begin{array}{lrcl} \Sigma: & S(x_1,x_2,x_3) & \rightarrow & T(x_1,x_2) \\ & S(x_1,x_2,x_3) & \rightarrow & T(x_2,x_3) \\ \Delta: & T(x_1,x_2) & \rightarrow & W(x_1,x_2,z) \end{array}$$

- Then substitute in the definition of W:

$$\begin{array}{rcl} S(x_1,x_2,y) & \rightarrow & W(x_1,x_2,z) \\ S(y,x_1,x_2) & \rightarrow & W(x_1,x_2,z) \end{array}$$

to get $\mathbf{\Sigma} \circ \mathbf{\Delta}$.

Composition: not without Skolem functions



Composition: not without equality

- Mapping Σ:

$$\text{Empl}(e) \quad \rightarrow \quad \text{Mngr}(e, m)$$

- Mapping Δ:

$$\begin{array}{rcl} \text{Mngr}(e, m) & \rightarrow & \text{Mngr'}(e, m) \\ \text{Mngr}(e, e) & \rightarrow & \text{SelfMng}(e) \end{array}$$

- Composition:

$$\begin{array}{rcl} \text{Empl}(e) & \rightarrow & \text{Mngr'}(e, f(e)) \\ \text{Empl}(e) \wedge e = f(e) & \rightarrow & \text{SelfMng}(e) \end{array}$$

Composable class of mappings

Mappings with Skolem functions and equality compose!

- Replace all nulls by Skolem functions:
  - $\text{Empl}(e) \;\rightarrow\; \text{Mngr}(e, m)$ becomes
    $\text{Empl}(e) \;\rightarrow\; \text{Mngr}(e, f(e))$
  - Δ stays as before
- Use substitution:
  - $\text{Mngr}(e, m) \;\rightarrow\; \text{Mngr'}(e, m)$ becomes
    $\text{Empl}(e) \;\rightarrow\; \text{Mngr'}(e, f(e))$
  - $\text{Mngr}(e, e) \;\rightarrow\; \text{SelfMng}(e)$ becomes
    $\text{Empl}(e) \wedge e = f(e) \;\rightarrow\; \text{SelfMng}(e)$

Complexity summery

- $(S, T) \in [\![\Sigma]\!]$: PTIME
  (easy, relational query evaluation)
- $(S, T) \in [\![\Sigma \circ \Gamma]\!]$: NP-complete
  (FKPT'04; improved examples of hardness in LS'08)
- certain answers:
  - undecidable for RA
  - PTIME for CQ

  (folklore)