

Mapping Maintenance in XML

P2P Databases

Dario Colazzo

LRI - Université Paris Sud

Carlo Sartiani

Dipartimento di Informatica - Università di Pisa

Motivating Example (1/3)

```
PisaBib = bib[(Author)*]
Author = author[Name, Affiliation, Paper*]
Name = name[String]
Affiliation = affiliation[String]
Paper = paper[Title, Year]
Title = title[String]
Year = year[Integer]
```

```
NYBib = bib[(Article|Book)*]
Article = article[Author*,Title,Year, RefCode]
Author = author[String]
Title = title[String]
Year = year[Integer]
Book = book[Author*,Title,Year, RefCode]
RefCode = refCode[String]
```



```
M:
...
for $aut in $input/author,
...
return author[$aut/name/text()]
...
```


Motivating Example (2/3)

```
for $aut in $bib/author,
```

```
...
```

```
where $aut/name = 'Mary F. Fernandez'
```

```
...
```

CORRECT!



```
for $aut in $bib/author,
```

```
...
```

```
where $aut = 'Mary F. Fernandez'
```

```
...
```


Motivating Example (3/3)

- p_j changes its schema

...

```
Author = author[first[String],second[String]]
```

...

- mapping M remains unchanged
- the reformulated query does not match the target schema

Introduction

- we want to study the maintenance of mappings in XML p2p databases
 - detecting when a mapping becomes corrupted due to changes in the peer schemas
- relevant for the data exchange problem too

Outline

- reference scenario
- mapping correctness
- correctness checking
- complexity remarks
- conclusions and future work

Reference Scenario

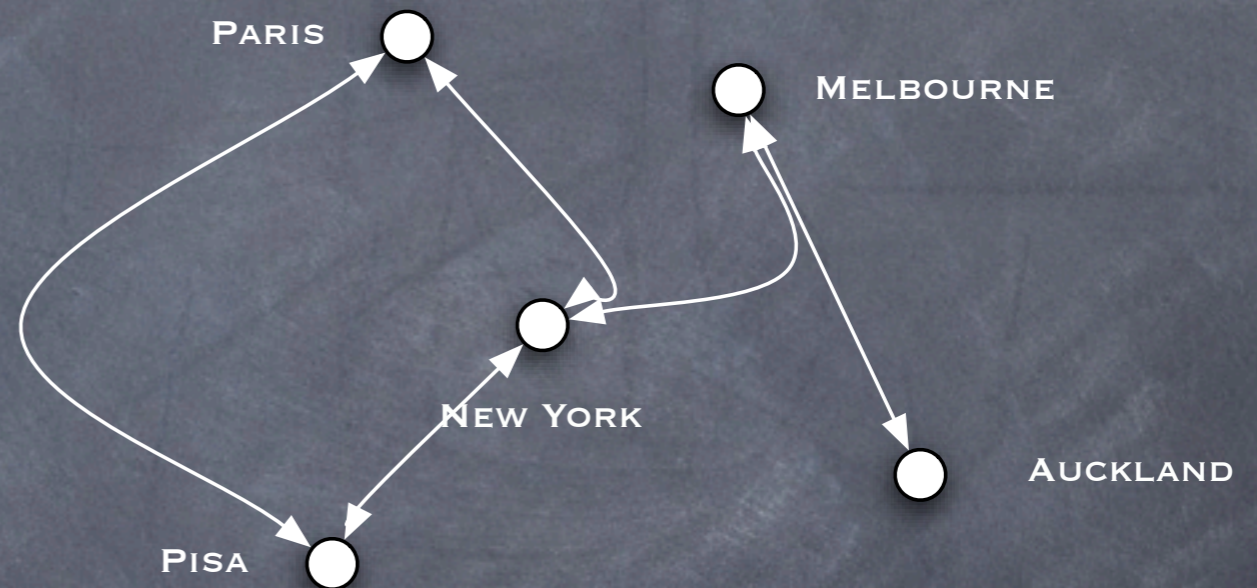
System Model

- an unstructured p2p system inspired by Piazza

- each peer comprises

- a local schema
- a peer view
- a set of neighbors
- a set of schema

mappings connecting its view to the views of its neighbors



Query Language

- simple query language inspired by XQuery (μ XQuery)
- for/let/where/return clauses
- simple predicate language
 - conjunction, disjunction, or negation of variable comparisons

Grammar

Q ::= $()$ | b | $l[Q]$ | Q, Q | \bar{x} child :: *NodeTest* | \bar{x} dos :: *NodeTest*
| for \bar{x} in Q return Q | let x ::= Q return Q
| for \bar{x} in Q where P return Q | let x ::= Q where P return Q

NodeTest ::= l | node() | text()

P ::= true | χ δ χ | empty(χ) | P or P | not P | (P)

χ ::= \bar{x} | x

δ ::= = | <

Semantics

$$\begin{array}{lll} \llbracket b \rrbracket_\rho & \triangleq b & \llbracket x \rrbracket_\rho \triangleq \rho(x) \\ \llbracket \bar{x} \rrbracket_\rho & \triangleq \rho(\bar{x}) & \llbracket () \rrbracket_\rho \triangleq () \\ \llbracket Q_1, Q_2 \rrbracket_\rho & \triangleq \llbracket Q_1 \rrbracket_\rho, \llbracket Q_2 \rrbracket_\rho & \llbracket l[Q] \rrbracket_\rho \triangleq l[\llbracket Q \rrbracket_\rho] \end{array}$$

$$\llbracket \bar{x} \text{ child} :: \text{NodeTest} \rrbracket_\rho \triangleq \text{childr}(\llbracket \bar{x} \rrbracket_\rho) :: \text{NodeTest}$$

$$\llbracket \bar{x} \text{ dos} :: \text{NodeTest} \rrbracket_\rho \triangleq \text{dos}(\llbracket \bar{x} \rrbracket_\rho) :: \text{NodeTest}$$

$$\llbracket \text{let } x ::= Q_1 \text{ return } Q_2 \rrbracket_\rho \triangleq \llbracket Q_2 \rrbracket_{\rho, x \mapsto \llbracket Q_1 \rrbracket_\rho}$$

$$\llbracket \text{for } \bar{x} \text{ in } Q_1 \text{ return } Q_2 \rrbracket_\rho \triangleq \prod_{t \in \text{trees}(\llbracket Q_1 \rrbracket_\rho)} \llbracket Q_2 \rrbracket_{\rho, \bar{x} \mapsto t}$$

$$\llbracket \text{let } x ::= Q_1 \text{ where } P \text{ return } Q_2 \rrbracket_\rho \triangleq \text{if } P(\rho, x \mapsto \llbracket Q_1 \rrbracket_\rho) \text{ then } \llbracket Q_2 \rrbracket_{\rho, x \mapsto \llbracket Q_1 \rrbracket_\rho} \text{ else } ()$$

$$\llbracket \text{for } \bar{x} \text{ in } Q_1 \text{ where } P \text{ return } Q_2 \rrbracket_\rho \triangleq \prod_{t \in \text{trees}(\llbracket Q_1 \rrbracket_\rho)} (\text{if } P(\rho, \bar{x} \mapsto t) \text{ then } \llbracket Q_2 \rrbracket_{\rho, \bar{x} \mapsto t} \text{ else } ())$$

Type Language

- inspired by XDuce
- a variation of the language presented in [ICFP04]
 - unordered types
 - no vertical recursion

$$T ::= () \mid B \mid l[T] \mid T, T \mid T \mid T \mid T^*$$
$$B ::= \text{String}$$

Mapping Correctness

Desiderata

- to capture
 - errors wrt the source schema
 - obsolete or dead rules
 - errors wrt the target schema
 - transforming a source instance into a fragment of a target schema instance
- independence from
 - the type system being used
 - query reformulation and answering

Mapping Validity

- a mapping $\omega = \{q_k\}_{k:V_i \rightarrow V_j}$ is valid iff, for each query q_k , q_k is **correct** wrt V_i
 - for each non-empty subquery q of q_k ,
 $\exists d:V_i$ such that $q(d) \neq ()$
- based on the query correctness notion of [ICFP04] and [PlanX05]
- it allows for the detection of obsolete rules
 - for \$aut in \$bib/author,
 - \$ssn in \$aut/ssn,
 - ...

Mapping Correctness

- a mapping $\omega = \{q_k\}_k: V_i \rightarrow V_j$ is correct iff,
 - $\forall q_k, \forall d_h:V_i, \exists d_l:V_j$, such that, $q_k(d_h) \lesssim d_l$
- based on the value projection relation \lesssim
- independent from the type system and the reformulation and answering algorithms

$()$	\lesssim	f	f_1, f_2	\lesssim	f_3, f_4	if $(f_1 \lesssim f_3 \wedge f_2 \lesssim f_4)$
b_1	\lesssim	b_2	f_1	\lesssim	f_3	if $\exists f_2 : f_1 \lesssim f_2 \wedge f_2 \lesssim f_3$
f	\lesssim	$f, ()$	$l[f_1]$	\lesssim	$l[f_2]$	if $f_1 \lesssim f_2$
f_1, f_2	\lesssim	f_2, f_1				

Mapping Correctness

Example (1/3)

```
PisaBib = bib[(Author)*]
Author = author[Name, Affiliation, Paper*]
Name = name[String]
Affiliation = affiliation[String]
Paper = paper[Title, Year]
Title = title[String]
Year = year[Integer]
```

source schema

```
NYBib = bib[(Article|Book)*]
Article = article[Author*, Title, Year, RefCode]
Author = author[first[String], second[String]]
Title = title[String]
Year = year[Integer]
Book = book[Author*, Title, Year, RefCode]
RefCode = refCode[String]
```

target schema

Mapping Correctness

Example (2/3)

```
mapping
NYBibliography <-
Q1($input): for $y in $input/year return $y
Q2($input): for $t in $input/title return $t
Q3($input): for $p in $input//paper,
             $t in $p/title
             return article[ Q2($p), Q1($p),
                             for $aut in $input/author,
                             $pap in $aut/paper
                             $title in $pap/title
                             where $title = $t
                             return author[$aut/name/text()]]
Q4($input): for $bib in /bib return bib[Q3($bib)]
```


Mapping Correctness

Example (3/3)

```
bib[
  author[name["Luca Cardelli"],
    affiliation["Microsoft Research"],
    ...
  ]
]
```

source instance

```
bib[
  article[...],
  ...,
  author["Luca Cardelli"]
  ...
]
```

transformed instance

- the transformed instance cannot be projected into any target instance
 - author name mismatch

Correctness Checking

Towards an Operational Notion

- the notion of mapping correctness is not operational
 - universal quantification on data instances
- two steps
 - switch to types
 - type projection

Type System

- a variant of those of [ICFP04] and [PlanX05]
- focused on type inference
 - rather precise thanks to type splitting
- upper bound property
 - for any well-formed Γ and query Q :
 - $\Gamma \vdash Q : U \wedge \rho \in R(\Gamma) \Rightarrow [[Q]]_\rho \in [[U]]$

Type Projection

- extension of the relational projection
- given two type T_1 and T_2 , we say that T_1 is a projection of T_2 ($T_1 \preceq T_2$) if and only if:
$$\forall d_1 : T_1 \exists d_2 : T_2 . d_1 \preceq d_2$$
- a form of injective simulation among types

Type Projection Completeness

- given a mapping $\omega = \{q_k\}_k$ from V_i to V_j , ω is correct if $\forall q_k . \Gamma \vdash q_k : T$ and $T \lesssim V_j$, where Γ is an environment obtained from V_i
- this theorem ties mapping correctness and type projection
- we need now a way to check for type projection!

Type Approximation

$$\begin{array}{lcl} ()^\triangleleft & \triangleq & () \\ l[T]^\triangleleft & \triangleq & l[T^\triangleleft]? \\ T, U^\triangleleft & \triangleq & T^\triangleleft, U^\triangleleft \end{array} \qquad \begin{array}{lcl} T | U^\triangleleft & \triangleq & T^\triangleleft | U^\triangleleft \\ B^\triangleleft & \triangleq & B? \\ T_*^\triangleleft & \triangleq & T^\triangleleft_* \end{array}$$

Type Projection As Subtyping

$$\bullet T \lesssim U \Leftrightarrow T < U^\dagger$$

• consequences

• type projection is decidable [DalZilio04]

• type projection can be checked by relying on subtype-checking algorithms

Complexity Remarks

Type Inference

- exponential
 - type splitting
- in most cases the algorithm is polynomial
 - *-guarded union types
 - fully specified paths

Type Projection

- [DalZilio04] proved that subtype-checking is super-exponential for a superset of our type language
 - unordered sequence types
 - union types
 - par types
 - negation types
 - repetition types

Differences

- our type language does not contain
 - par types
 - negation
- much simpler

Complexity Claim

- type projection for our type language is supposed to be polynomial
- projection for union and product types is equivalent to the Wombat Eating Assignment (WEA) problem [GJ79]
 - can be efficiently solved by using 0-1 maximum flow algorithms for bipartite graphs
 - $O(nm(n+m)^3)$

Conclusions and Future Work

Conclusions

- we defined a “semantic” notion of schema mapping correctness
 - independent from the type system
 - independent from query reformulation and answering algorithms
- we related correctness to type projection, as well as type projection to subtyping
- we proved decidability and complexity of type projection

Future Work

First Direction



- constraints
- M is correct if it is correct wrt S_i and S_j , and if $M(C_i)$ is "compatible" with C_j

Future Work

Second Direction

- recursive types
 - type projection as subtyping should hold in the presence of recursive types too
 - recursive type as (infinite) union of non-recursive types

Future Work

Third Direction

- improving type inference precision
 - less false negatives

References

- [ICFP04] Colazzo, D., Ghelli, G., Manghi, P., Sartiani, C.: Types for Path Correctness of XML Queries. In: Proceedings of the 2004 International Conference on Functional Programming (ICFP), Snowbird, Utah, September 19–22, 2004. (2004)
- [PlanX05] Colazzo, D., Sartiani, C.: Typechecking Queries for Maintaining Schema Mappings in XML P2P Databases. In: Proceedings of the 3th Workshop on Programming Language Technologies for XML (Plan-X), in conjunction with POPL 2005.
- [DalZilio04] Dal-Zilio, S., Lugiez, D., Meyssonier, C.: A logic you can count on. In Jones, N.D., Leroy, X., eds.: POPL, ACM (2004) 135–146
- [GJ79] Michael Garey and David Johnson: Computers and Intractability - A Guide to the Theory of NP-completeness; Freeman, 1979