Representing and Querying XML with Incomplete Information

Serge Abiteboul
INRIA

Joint work with Victor Vianu, UCSD and Luc Segoufin, INRIA
Organization

• Incomplete databases
• XML
• Motivations
• Setting: documents, types, queries
• Example
• What can be done: PTIME
• Limits: exponential blow-up
Incomplete databases
Incomplete databases

• Moshe Vardi from Rice is presenting a paper, but I don’t know the title
  – Is Moshe from U. Wisconsin? No
  – Is Moshe presenting a paper? Yes
  – Is Moshe speaking on the usual effectiveness of logic in computer science? Maybe

• My knowledge of the world is incomplete
Incomplete databases

• Moshe is from Rice and he is presenting a paper
  – Is Moshe from U. Wisconsin? No
  – Is Moshe presenting a paper? Yes
  – Is Moshe speaking on the unusual effectiveness of logic in computer science? Maybe

• My knowledge of the world is incomplete

• Natural for logicians – an early discovery for database people
Incomplete databases

• An old story
• Main idea: database is the set of all possible worlds – incompleteness comes from the fact that we do not know which one it is
• Sure answers: true in all worlds
• Possible answers: possible in some worlds
Database people look at simple models

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>@</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$2</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$2</td>
<td>6</td>
<td>@</td>
<td></td>
</tr>
</tbody>
</table>
Incomplete databases

• From the very first papers of Codd
  • Tables = simple descriptions of the possible words
    – Landmark paper: Imielinski and Lipski – notion of representation system
• Logical approach
  – knowledge of the world = a set of logical sentences
  – Landmark paper: Vardi’s *Querying Logical Databases* and the complexity of nulls
• Results on meaning of answers and complexity
Representation system

- A system to represent the information we have – $T$
- Need a representation of (possible) answers – $q(T)$

\[
\begin{align*}
\text{rep}(T) & \xrightarrow{q} q(\text{rep}(T)) = \text{rep}(q(T)) \\
T & \xrightarrow{q} q(T)
\end{align*}
\]

- This is a (weak) **representation system**
- You may think of it as any limited logic with the property of the diagram
XML
XML

• New fashion: standard for the web to replace HTML

• Data exchange model: semistructured data

• Bottom line: trees with tagged vertices

• Query languages: mix of information retrieval and relational query languages
Review of XML and DTDs

<dealer>
  <UsedCars>
    <ad>
      <model>Honda</model>
      <year>96</year>
    </ad>
  </UsedCars>
  <NewCars>
    <ad>
      <model>Acura</model>
    </ad>
  </NewCars>
</dealer>
Typing XML: Data Type Definition

\[ \Sigma: \text{alphabet of element names, } \text{root} \in \Sigma \]

set of rules:

\[ e \rightarrow r \]

element name \[ \rightarrow \]

regular expression over \(\Sigma\)
Documents satisfying a DTD

Set of trees satisfying DTD \( d \): \( \text{Tree}(d) \)
Example

A DTD and a tree satisfying it:

```
root     section*;
section     intro, section*, conclusions;
```

```
root
  └── section
      └── intro
      └── section
          └── intro
              └── section
                  └── intro
                      └── conc
```
Shortcoming of DTDs: context-free definition

Cannot write a DTD describing this single document: ad has different structure in different contexts
Solution: *specialization (decoupled tags)*

specialize \( \text{ad} \): \( \text{ad}^{\text{used}} \) and \( \text{ad}^{\text{new}} \)

- Dealer \( \longrightarrow \) UsedCars, NewCars
- UsedCars \( \longrightarrow \) \( \text{ad}^{\text{used}} \)
- NewCars \( \longrightarrow \) \( \text{ad}^{\text{new}} \)
- \( \text{ad}^{\text{used}} \) \( \longrightarrow \) model, year
- \( \text{ad}^{\text{new}} \) \( \longrightarrow \) model

\[
\begin{align*}
\text{h}(\text{ad}^{\text{new}}) &= \text{ad} \\
\text{h}(\text{ad}^{\text{used}}) &= \text{ad}
\end{align*}
\]
What sets of trees can specialized DTDs define?

 Exactly the regular tree languages of unranked trees!
[Bruggemann-Klein+Murata+Wood]

Consequences:

same closure properties (e.g. intersection, union, complement) and complexities of manipulations
Motivations
Massive repository of XML documents

- Motivated by the web and the Xyleme project at INRIA
- Objectives:
  - store massive volumes of XML documents
  - provide queries and other services (monitoring, integration, classification, web crawling…)
  - emphasis on change management
Why incomplete information?

• Information in such a warehouse is seldom complete:
  – limited storage capacity
  – data change
  – expired data
  – unavailable data (server down, not proper access rights), etc

• This work: simple, practically appealing approach to managing incomplete information in this context
Incomplete Information

Scenario:

• Information continuously enriched by successive queries to XML sources

• Need to:
  – represent incomplete information
  – intelligently answer queries
  – using incomplete information
Wish list:

- Need some form of representation system
- Efficient incremental maintenance through consecutive queries

\[
\text{rep}(T_k) = \text{rep}(T_0) \cap q_1^{-1}(A_1) \cap \ldots \cap q_k^{-1}(A_k)
\]
Useful also for queries

- Compute a representation of the answer

\[
\begin{align*}
\text{rep}(T) & \xrightarrow{\text{q}} \text{q}(\text{rep}(T)) = \text{rep}(\text{q}(T)) \\
\text{rep} & \uparrow \quad \text{rep} \\
T & \xrightarrow{\text{q}} \text{q}(T)
\end{align*}
\]

Need: strong representation system
wrt query language
Answer queries – use what you have

• Decide if available info is enough to fully answer the query
  – similar to answering queries using views

• Answering queries using views: heavily studied problem
  – V1/Q1, V2/Q2,…, Vn/Qn
  – Comes a query Q
  – Can it be answered fully, partial answer…
Answer queries – Ask for more

• If it is not sufficient, seek additional information
  – mediator problem: find “minimal” set of additional queries to sources needed to fully answer query
  – use representation of incomplete info to guide mediator
Challenge: balance expressiveness and tractability!

- Choice of: XML document types (DTDs)
  Query language

- This work: one proposal
  - simple, practically appealing, many limitations

- Justification:
  - extra features lead to serious problems!
Setting
Documents and Types

- XML abstraction: unranked trees with labels and values
- simplified DTD: unordered, simple cardinality constraints

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>price</th>
<th>category</th>
<th>picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

1: exactly one
+: at least one
*: unrestricted
?: zero or one
Query Language

- prefix-selection queries (ps-queries)

No branching: siblings with same label
Prefix-selection Queries

Find electronics products with price < 200 and without pictures
- display all info about their name, the price, the category and subcategory)

Important assumption:
- persistent node ids!
- queries return actual nodes from the input: can “join” answers from consecutive queries
Example
Source DTD (tree type)

```
catalog
  +
  product
    1
    name  1
    price  1
    cat
    picture
    1
    subcat
```

Query 1

```
catalog
  product
    name  price<200  cat=elec
    subcat
```

Answer to Query 1

```
catalog
  product
    Canon 120  elec  camera
  product
    Nikon 199  elec  camera
  product
    Sony 175  elec  cd-player
```
Incomplete information after Query 1

known information
prefix data tree
Incomplete information after Query 1

known information
prefix data tree

missing information
Incomplete information after Query 1

known information
prefix data tree

missing information
extended tree type:
--conditions on data values
--specialization, disjunction
Incomplete information after Query 1

known information
prefix data tree

missing information
extended tree type:
--conditions on data values
--specialization, disjunction

Incomplete tree
Incomplete tree
Incomplete tree

type: $t_1 \lor \ldots \lor t_n$
Query 2

catalog
    product
        name
cat=elec
date
    picture
    subcat=camera

Answer to Query 2

catalog
    product
        product
            Canon
cat=elec
date
camera
c.jpg

    product
            Olympus
cat=elec
    camera
c.jpg
Incomplete tree after Query 2
Suppose next query is:

**Query 3:** find the name, price and pictures of all cameras costing less than $200 and having at least one picture

Can be fully answered using available information

Note: need tests of the form $q(rep(T)) = \Phi$
Query 4: find all cameras

Using available information can:

--provide the complete list of cameras that are less than $200 or have a picture;
--tell the user that there may be more cameras (that are expensive and have no pictures).

Can fully answer query by asking the extra query:

Query 5: find the cameras that cost at least $200 and have no picture.
Measuring up to the wish list

- Incomplete trees can be incrementally computed in PTIME

\[ \text{rep}(T') = \text{rep}(T) \cap q^{-1}(A) \]
Strong representation system for ps-queries

\[ \text{rep}(T) \xrightarrow{q} q(\text{rep}(T)) = \text{rep}(q(T)) \]

T \xrightarrow{\text{PTIME}} q \xrightarrow{q} q(T)

reachable incomplete tree

incomplete tree for possible answers
• Can check in PTIME if a ps-query can be fully answered with incomplete information described by incomplete tree

Note: in terms of answering queries using views: can \( q \) be answered using views provided by \( q_1, \ldots, q_k \)

• Can complete answer: given \( q \) and \( T \), generate in PTIME a non-redundant set of queries sufficient to fully answer \( q \)

Non-redundant:
-- no existing nodes are retrieved again
-- no queries are asked that always have empty answer on the possible inputs

• Key fact: can check in PTIME if \( q(\text{rep}(T)) = \Phi \)
Exponential blow-up
Drawback: incomplete tree can become exponential wrt overall query/answer sequence

Input tree = \[ T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow \ldots \rightarrow T_k \]

\[(q_1, A_1) (q_2, A_2) (q_3, A_3) \ldots (q_k, A_k)\]

Example

\[\begin{array}{c}
q_i \\
\text{root}
\end{array}\]

\[\begin{array}{c}
1 \\
a \\
b
1
\end{array}\]

\[\begin{array}{cc}
a = i & b = i
\end{array}\]

Basic trade-off: conciseness vs. efficiency of manipulations

NP-complete whether \( t \) is possible prefix for inputs compatible with \((q_i, A_i)\) and \(T_0\)
Dealing with the exponential blowup

- **augment incomplete trees**: conjunctions of disjunctions of types

**Example**

```
\[
\begin{array}{c}
q_i \\
\text{root} \\
\text{empty answers}
\end{array}
\]
```

```
\[
\begin{array}{cc}
\text{input type:} & \text{root} \\
1 & 1 \\
a & b
\end{array}
\]
```

```
\[
\text{root: } (a_1 \lor b_1) \land \ldots \land (a_n \lor b_n)
\]
```

where

```
\[
\begin{array}{cc}
a_i : a \neq i & b_i : b \neq i
\end{array}
\]
```

--size of incomplete tree stays polynomial

--increase in complexity of manipulations
Dealing with the exponential blowup

- restrict tree types and ps-queries
  --non-recursive tree types
  --ps-queries testing data values only along one path

```
catalog
   product
      name  cat=elec
      |
      subcat=camera
```

--size of incomplete tree stays polynomial
--no increase in complexity of manipulations
Dealing with the exponential blowup

• heuristics to deal with large incomplete trees

  --ask linear set of **additional queries** (always possible)
  can guarantee overall polynomial size

  --**gracefully loose** some of the information
  to shrink the incomplete tree (e.g., “un-specialize” types)
Extensions of query language yield problems:

- No user-friendly representation of incomplete info
- No strong representation system
- High complexity for basic manipulations
- Undecidability of basic questions
Branching

- Incomplete trees remain strong representation system
- Can be maintained incrementally in PTIME
- But: $q(T)$ exponential in $T$

```
input type       incomplete tree $T$       query $q$

root
*  |  
  a
*  |  
b

root
a1 a2 .... an

root
a a .... a

b=1 b=2 .... b=n

specializations of a

n! possibilities…
```
Branching + constructed answers

- no (known) strong representation system

\[
\begin{array}{cc}
\text{where} & \text{construct} \\
\text{root} & \text{root} \\
a & b: f(X) \\
a & c: g(Y) \\
X & \text{output: } b^n c^n \\
Y & \\
\end{array}
\]

Note: connection to type inference
Branching + optional subtrees

Find all cameras and their pictures (if any)

![Tree Diagram]

**Complexity**: given tree type $T$, query/answer $<q, A>$, new query $q'$

$t$ is a sure prefix for $q'(\text{rep}(T) \cap q^{-1}(A))$ is co-NP-complete

*There can be no strong representation system for which incremental maintenance, computing $q(T)$, and testing that $t$ is a sure prefix of rep($T$) are all in PTIME*
Very powerful transformers: k-pebble transducers with data value selections

-Milo, Suciu, V.

• subsume tree manipulation core of XML-QL and XSL

• k-pebble automata provide representation system, can be maintained in PTIME wrt overall sequence

• no (known) strong representation system

• testing \( \text{rep}(T) = \Phi \) is non-elementary!
Branching + join on data values + negation

Comparisons of data values, negative sub-trees

Basic questions are undecidable:

Given: input tree type $T$
query/answers $<q_1,A_1> \ldots <q_n,A_n>$
query $q$

Test whether $q$ is always empty on compatible inputs

*There cannot exist an effective strong representation system for which such questions are decidable*
Similar:

Branching + join on data values
+ optional sub-trees + construction

Trade-off between negation and optional sub-trees + construction

Recursive path expressions and join on data values

Proofs: reduce implication of FDs + INCDs,
emptiness of intersection of two CFGs
Other issues

• Persistent node id assumption
  • without it approach still works, but weaker
  • cannot enrich info about nodes

• Order: input tree, input DTD, used by queries

New problems:
q₁: list all a elements
q₂: list all b elements
q₃: list all elements  -- can answer using q₁,q₂?
  yes: if input DTD is a* b*
  no:  if input DTD is (a+b)*
Conclusion

• Simple framework for acquiring, maintaining, and querying incomplete XML documents
• Answer queries as best possible given incomplete information, or generate additional non-redundant queries to provide full answers
• Limitations: simple DTDs and queries, persistent id assumption, no order
• But: even small extensions lead to serious problems