A Normal Form for XML Documents

Marcelo Arenas
Department of Computer Science
University of Toronto
marenas@cs.toronto.edu

Joint work with Leonid Libkin
Outline

- Motivation
- Functional dependencies for XML
- A normal form for XML documents
- An algorithm for normalizing XML documents
- Implication problem for functional dependencies
- Ongoing work
XML Data: University Database

A Normal Form for XML Documents 2
<!DOCTYPE courses [
  <!ELEMENT courses (course*)>
  <!ELEMENT course (title, taken_by)>
  <!ATTLIST course
cno CDATA #REQUIRED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT taken_by (student*)>
  <!ELEMENT student (name, grade)>
  <!ATTLIST student
sno CDATA #REQUIRED>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT grade (#PCDATA)>
]>
Semantic Information

■ XML specifications can include semantic information:
  ◇ Two distinct course elements cannot have the same cno
  ◇ Two distinct student subelements of the same course cannot have the same sno
  ◇ Two student elements with the same sno value must have the same name

■ XML documents can contain redundant information
Our Goal: Schema Refinement

A Normal Form for XML Documents
Relational Ideas can be Inadequate: DBLP
Problems to Address

- Definition of a language for expressing functional dependencies
  - Simple language
  - Absolute and Relative constraints
  - The order of children in an XML tree is irrelevant as far as satisfaction of constraints is concerned

- Definition of a normal form for XML documents (XNF)

- Construction of an algorithm for normalizing XML documents
  - Implication problem for functional dependencies
  - Testing if a specification is in XNF
Framework: DTDs

We do not consider mixed content, IDs and IDREFs

Notation:

- \textit{Paths}(D): all paths in a DTD \(D\)
  
  In the university database example:

  \begin{align*}
  \text{courses} & \quad \text{courses.course} \\
  \text{courses.course.@cno} & \quad \text{courses.course.title} \\
  \text{courses.course.title.$} & \quad \ldots
  \end{align*}

- \textit{EPaths}(D): all paths that end with an element type

  \text{courses.course} \text{ is in } \text{EPaths}(D)

  \text{courses.course.@cno} \text{ is not}
Framework: XML Trees

A Normal Form for XML Documents
Tree Tuples

A tree tuple $t$ in a DTD $D$ is a function from $\text{Paths}(D)$ to $\text{Vertices} \cup \text{Strings} \cup \{\perp\}$

$t(\text{courses}) = v_0$
$t(\text{courses.course}) = v_1$
$t(\text{courses.course.@cno}) = \text{csc200}$
$t(\text{courses.course.title}) = v_2$
$t(\text{courses.course.title.S}) = \text{Automata Theory}$
$t(p) = \perp$, for the remaining paths
XML tree: Set of Tree Tuples

- An XML tree can be represented as a set of tree tuples, if we consider it as an unordered tree.

- We consider tuples containing a maximal amount of information (minimal set of ⊥ values).

- If $T$ is the XML tree containing information about courses, then $Tuples_D(T) = \{t_1, t_2, t_3, t_4\}$, where ...
Tree Tuples of $T$

$t_1: v_0$

$v_1$

@cno
"csc200" $v_2$

S

"Automata Theory" $v_4$

@sno
"st1" $v_5$

S

"Deere" "A+"

$t_2: v_0$

$v_1$

@cno
"csc200" $v_2$

S

"Automata Theory" $v_7$

@sno
"st2" $v_8$

S

"Smith" "B-"

$t_3: \ldots$
Functional Dependencies for XML

- A functional dependency over a DTD $D$ is an expression
  \[ S_1 \rightarrow S_2 \]
  where $S_1$ and $S_2$ are finite subsets of $\text{Paths}(D)$

- $T \models S_1 \rightarrow S_2$ if for every $t_1, t_2 \in Tuples_D(T)$,
  \[ t_1.S_1 = t_2.S_1 \text{ and } t_1.S_1 \neq \bot \implies t_1.S_2 = t_2.S_2. \]
Back to the University Example

- Two distinct course elements cannot have the same cno:

  \[ \text{courses.course.@cno} \rightarrow \text{courses.course} \]

- Two distinct student subelements of the same course cannot have the same sno:

  \[ \{\text{courses.course}, \text{courses.course.taken_by.student.@sno}\} \rightarrow \text{courses.course.taken_by.student} \]

- Two student elements with the same sno value must have the same name:

  \[ \text{courses.course.taken_by.student.@sno} \rightarrow \text{courses.course.taken_by.student.name.S.} \]
XNF: An XML Normal Form

- XML specification: a DTD $D$ and a set of functional dependencies $\Sigma$

- $(D, \Sigma)$ is in XML Normal Form (XNF) if:
  
  for every non-trivial functional dependency $\varphi \in (D, \Sigma)^+$ of the form $S \rightarrow p.@l$ or $S \rightarrow p.S$, it is the case that $S \rightarrow p$ is in $(D, \Sigma)^+$

- XNF generalizes BCNF and a normal form for nested relations (NNF) when those are coded as XML documents
XNF: Back to the Examples

- University specification is not in XNF:

  $$\text{courses.course.taken_by.student.@sno} \rightarrow$$

  $$\text{courses.course.taken_by.student.name}$$

  is not in $$(D, \Sigma)^+$$.

- DBLP specification is not in XNF:

  $$\text{db.conf.issue} \rightarrow \text{db.conf.issue.inproceedings.@year} \in (D, \Sigma)^+$$

  $$\text{db.conf.issue} \rightarrow \text{db.conf.issue.inproceedings} \notin (D, \Sigma)^+$$

- Proposed solutions are in XNF
Normalizing XML Documents

We consider functional dependencies of the form

$$\{q, \ p_1.@l_1, \ldots, p_n.@l_n\} \rightarrow p$$

where $n \geq 0$, $q \in EPaths(D)$ and $p \in Paths(D)$

The normalization algorithm applies two transformations until the schema is in XNF:

- **Moving attributes**: if there is an anomalous functional dependency $q \rightarrow p.@l$ in $(D, \Sigma)^+$, then

\[ r \]
\[ \text{p} \]
\[ \text{q} \]
\[ \text{@l} \]
\[ \text{@m} \]
• Creating new element types: choose a minimal anomalous functional dependency \( \{q, p_1.@l_1, \ldots, p_n.@l_n\} \rightarrow p.@l \) and
Normalizing XML Documents

- **Theorem** *The decomposition algorithm terminates and outputs a specification in XNF*

- Our transformations do not lose information: there are XQuery queries that translate back and forth two schemas (a’ la Hull’s information capacity of schemas)

- It involves implication of functional dependencies
Reasoning about Functional Dependencies

- Typically, regular expressions used in DTDs are rather simple

- *D is a simple DTD* if *D* contains regular expression of the form \(s_1, \ldots, s_n\), where
  - each \(s_i\) is either one of \(a_i, a_i?, a_i^+\) or \(a_i^*\)
  - for \(i \neq j\), \(a_i \neq a_j\)

  \(D\) can also contain “permutations” of this type of expressions: 
  \((course \mid info)^*\)
Example: ebXML

Business process specification schema of ebXML:

```xml
<!ELEMENT ProcessSpecification (Documentation*, SubstitutionSet*, (Include |
    BusinessTransaction | MultiPartyCollaboration)*)>
<!ELEMENT Include (Documentation*)>
<!ELEMENT BusinessDocument (ConditionExpression?, Documentation*)>
<!ELEMENT SubstitutionSet (DocumentSubstitution | AttributeSubstitution |
    Documentation)>
<!ELEMENT BinaryCollaboration (Documentation*, InitiatingRole, RespondingRole, |
    (Documentation | Start | Transition | Success | Failure | |
    BusinessTransactionActivity | CollaborationActivity | Fork | Join)*)>
<!ELEMENT Transition (ConditionExpression?, Documentation*)>
```
Reasoning about Functional Dependencies

- **Theorem** *For simple DTDs*
  - The implication problem for FDs is solvable in quadratic time
  - Testing if a specification is in XNF can be done in cubic time

- **Other results**
  - There is a larger class of DTDs for which these problems are tractable ("small" number of disjunctions)
  - There is a class of DTDs for which these problems are coNP-complete
Ongoing Work

- Improve the decomposition algorithm in various ways
- Find a complete classification of the complexity of the implication problem for various classes of DTDs
- Construct a more expressive language for functional dependencies (regular expressions)
- Consider other anomalies and other integrity constraints
- Implementation