Similarity of XML Schema Definitions

Irena Mlynkova
irena.mlynkova@mff.cuni.cz

Charles University
Faculty of Mathematics and Physics
Department of Software Engineering
Prague, Czech Republic
Introduction

• **XML** = a standard for data representation and manipulation
  ⇒ used in most areas of IT
  • Clustering, dissemination-based applications, schema integration systems, data warehousing, e-commerce, semantic query processing, ...

• **Our focus: similarity of XML schemas**
  • Quantitative = the degree of difference of the schemas
  • Qualitative = how the schemas relate
    • E.g. which of the schemas is more general
Goals of the Paper

• **Disadvantages to be solved:**
  • Current approaches focus on
    • Semantic similarity
    • Similarity of DTDs
  • Structural similarity is analyzed trivially
    • Comparison of leaf nodes / direct child nodes

• **Our aims:**
  • Focus on of XML Schema constructs
    • Structural and semantic equivalence
  • Emphasis on structural similarity
    • Utilized edit distance
  • Preservation of exploitation of semantic similarity
Equivalence of XSD Constructs

• XML Schema constructs: lot of “syntactic sugar”

Definition. Let $S_x$ and $S_y$ be XSD fragments. Let $I(S) = \{D \text{ s.t. } D \text{ is an XML document fragment valid against } S\}$.

• $S_x$ and $S_y$ are **structurally equivalent**, $S_x \sim S_y$, if $I(S_x) = I(S_y)$.

• $S_x$ and $S_y$ are **semantically equivalent**, $S_x \approx S_y$, if they abstract the same reality.

• A vague definition

$\implies$ Having a set $X$ of all XSD constructs:

• Quotient sets $X/\sim$ and $X/\approx$, respective equivalence classes, canonical representatives
# Equivalence

## Classes of $\sim$

<table>
<thead>
<tr>
<th>Class</th>
<th>Constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{ST}$</td>
<td>globally defined simple type, locally defined simple type</td>
</tr>
<tr>
<td>$C_{CT}$</td>
<td>globally defined complex type, locally defined complex type</td>
</tr>
<tr>
<td>$C_{El}$</td>
<td>referenced element, locally defined element</td>
</tr>
<tr>
<td>$C_{At}$</td>
<td>referenced attribute, locally defined attribute, attribute referenced via an attribute group</td>
</tr>
<tr>
<td>$C_{ElGr}$</td>
<td>content model referenced via an element group, locally defined content model</td>
</tr>
<tr>
<td>$C_{Seq}$</td>
<td>unordered sequence of elements $e_1, e_2, ..., e_l$, choice of all possible ordered sequences of $e_1, e_2, ..., e_l$</td>
</tr>
<tr>
<td>$C_{CDe}$</td>
<td>derived complex type, newly defined complex type</td>
</tr>
<tr>
<td>$C_{SubSk}$</td>
<td>elements in a substitution group $G$, choice of elements in $G$</td>
</tr>
<tr>
<td>$C_{Sub}$</td>
<td>data types $M_1, M_2, ..., M_k$ derived from type $M$, choice of content models defined in $M_1, M_2, ..., M_k, M$</td>
</tr>
</tbody>
</table>

### Canonical representative

- locally defined simple type
- locally defined complex type
- locally defined element
- locally defined attribute
- locally defined content model
- choice of all possible ordered sequences of $e_1, e_2, ..., e_l$
- newly defined complex type
- choice of elements in $G$
- choice of content models defined in $M_1, M_2, ..., M_k, M$
Examples
### Equivalence

**Classes of \( \approx \)**

<table>
<thead>
<tr>
<th>Class</th>
<th>Constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C'_{IdRef} )</td>
<td>locally defined schema fragment, schema fragment referenced via IDREF attribute</td>
</tr>
<tr>
<td>( C'_{KeyRef} )</td>
<td>locally defined schema fragment, schema fragment referenced via keyref element</td>
</tr>
</tbody>
</table>

**Example**

```xml
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID"/>
  </xs:complexType>
</xs:element>

<xs:element name="relationships">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="personInferior" maxOccurs="unbounded">
        <xs:element name="name" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
    </xs:complexType>
  </xs:element>
</xs:element>
```
Similarity Evaluation

- Similarity of XML documents = tree edit distance
  - XML documents $D_A$ and $D_B$ = labelled trees $T_A$ and $T_B$
  - Number of operations to transform $T_A$ to $T_B$
- Basic tree edit operations: Relabeling, InsertNode, DeleteNode
  - XML data: sharing, repetitions, recursion, …
    $\Rightarrow$ XML tree edit operations: InsertTree, DeleteTree
- Algorithm:
  1. XSDs are parsed + their trees are constructed
  2. Costs for inserting/deleting subtrees are computed
  3. Resulting minimal edit distance is evaluated
    - Dynamic programming
XSD Tree Construction (1)

• XSD content models can be complex
  • “Syntactic sugar”, operators, recursion, shared fragments, ….

1. Normalization:
  • Replace each non-canonical construct with respective canonical representative of ~ and ≈
  • For each XSD construct \( v \) keep the set \( v_{eq~} \) and \( v_{eq≈} \) of classes it originally belonged to

⇒ Schema involves elements, attributes, operators choice and sequence, allowed occurrences, simple types and assertions
  • No shared schema fragments
  • Note: We omit solution of recursion for paper length
XSD Tree Construction (2)

2. **Simplification rules:**

| I.a) $(e_1 | e_2)^* \rightarrow e_1^*, e_2^*$ | II.a) $e_1^{++} \rightarrow e_1^+$ |
| I.b) $(e_1, e_2)^* \rightarrow e_1^*, e_2^*$ | II.b) $e_1^{**} \rightarrow e_1^*$ |
| I.c) $(e_1, e_2)? \rightarrow e_1?, e_2?$ | II.c) $e_1^*? \rightarrow e_1^*$ |
| I.d) $(e_1, e_2)^+ \rightarrow e_1^+, e_2^+$ | II.d) $e_1^*?* \rightarrow e_1^*$ |
| I.e) $(e_1 | e_2) \rightarrow e_1?, e_2?$ | II.e) $e_1^{**} \rightarrow e_1^*$ |
| | II.f) $e_1^{**} \rightarrow e_1^*$ |
| | II.g) $e_1^*+ \rightarrow e_1^*$ |
| | II.h) $e_1^*? \rightarrow e_1^*$ |
| | II.i) $e_1?? \rightarrow e_1?$ |

⇒ Cardinality constraints are connected to single elements, no usage of | (choice) operator
- A slight information loss
Example:

type

- element, “name”, [1,1], \{C_{El}\}, {}
  
  - string, “”, [], {}, {}
  
  - element, “first”, [0,1], \{C_{Seq}\}, {}
  
  - element, “surname”, [0,1], \{C_{Seq}\}, {}

label

- element, “person”, [1,*], \{C_{El}\}, {}
  
  - element, “email”, [0,*], {}, {}
  
  - string, ““, [], {}, {}

cardinality

- element, “employees”, [1,1], {}, {}

≈ classes

- element, “first”, [0,1], \{C_{Seq}\}, {}

~ classes

- element, “surname”, [0,1], \{C_{Seq}\}, {}
Tree Edit Operations

• Same as for XML trees: Relabeling, InsertNode, DeleteNode, InsertTree, DeleteTree
• Transformation of $T_A$ to $T_B$: various sequences of operations
• Optimization: allowable sequences
  • Tree $T$ may be inserted only if tree similar to $T$ occurs in $T_B$
  • Tree $T$ may be deleted only if tree similar to $T$ occurs in $T_A$
  • Tree that has been inserted via the InsertTree may not subsequently have additional nodes inserted
  • Tree that has been deleted via the DeleteTree may not previously have had nodes deleted
$Sim(v, v')$

\[ = \text{Max}(\text{SemanticSim}(v, v'), \text{SyntacticSim}(v, v')) \times \alpha_1 + \text{CardSim}(v, v') \times \alpha_2 + \text{StrFragSim}(v, v') \times \alpha_3 + \text{SemFragSim}(v, v') \times \alpha_4 + \text{DataTypeSim}(v, v') \times \alpha_5 \]

where \( \sum_{i=1}^{5} \alpha_i = 1 \) and \( \forall i : \alpha_i \geq 0 \).

**CardSim**

\[ = 0 \quad ; (v_{up} < v'_{low}) \lor (v'_{up} < v_{low}) \]
\[ = 1 \quad ; v_{up}, v'_{up} = \infty \land v_{low} = v'_{low} \]
\[ = 0.9 \quad ; v_{up}, v'_{up} = \infty \land v_{low} \neq v'_{low} \]
\[ = 0.6 \quad ; v_{up} = \infty \lor v'_{up} = \infty \]
\[ = \frac{\min(v_{up}, v'_{up}) - \max(v_{low}, v'_{low})}{\max(v_{up}, v'_{up}) - \min(v_{low}, v_{low})} \quad ; \text{otherwise} \]

**StrFragSim**

\[ = 1 \quad ; v_{eq}, v'_{eq} = \emptyset \]
\[ = \frac{|v_{eq} \cap v'_{eq}|}{|v_{eq} \cup v'_{eq}|} \quad ; \text{otherwise} \]

**SemFragSim**

\[ = 1 \quad ; v_{eq}, v'_{eq} = \emptyset \]
\[ = \frac{|v_{eq} \cap v'_{eq}|}{|v_{eq} \cup v'_{eq}|} \quad ; \text{otherwise} \]
Cost of Tree Edit Operations

- Inserting/deleting tree $T$:
  - Single InsertTree/DeleteTree … a combination of InsertTree/DeleteTree and Insert/Delete
  - Which is the best?
- Idea:
  - Pre-computed: $\text{Cost}_{\text{Graft}}(T), \text{Cost}_{\text{Prune}}(T)$ for each subtree $T$
  - Dynamic programming: finds the optimal sequence of edit operations
- Classical approach for tree edit distance
  - See the paper for details…
Experiments

- Testing set: 3 synthetic XSDs
  - I and II differ within ~, III differs in more aspects
  - Test A = we ignore the information on original XSD constructs
  - Test B = similarity is influenced by structural difference between XSD constructs
    - More precise results
  - Test C = structural differences are ignored
    - The same trend as in A, more precise
  - Test D, E = exploitation of SemanticSim
    - Expensive operation
    - Provides more precise results

<table>
<thead>
<tr>
<th>Test</th>
<th>$\alpha_3 = \alpha_4 = 0$</th>
<th>I × II</th>
<th>II × III</th>
<th>III × I</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>1.00</td>
<td>0.82</td>
<td>0.82</td>
</tr>
<tr>
<td>B</td>
<td>$\alpha_4 = 0$, $\alpha_3 \neq 0$</td>
<td>0.89</td>
<td>0.70</td>
<td>0.66</td>
</tr>
<tr>
<td>C</td>
<td>$\alpha_3 = 0$, $\alpha_4 \neq 0$</td>
<td>1.00</td>
<td>0.80</td>
<td>0.80</td>
</tr>
<tr>
<td>D</td>
<td>A without SemanticSim</td>
<td>1.00</td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td>E</td>
<td>B without SemanticSim</td>
<td>0.89</td>
<td>0.255</td>
<td>0.24</td>
</tr>
</tbody>
</table>
Conclusion

- Algorithm for evaluating XSD similarity
  - Emphasis on structural level
  - Coping with “syntactic sugar” of XML Schema
  - Exploitation of semantics
- Key idea: Combination of edit distance and semantic similarity
- Future work:
  - More elaborate testing
  - Other edit operations
    - Moving a node or adding/deleting a non-leaf node
  - Setting weights