Towards Inference of More Realistic XSDs

Irena Mlýnková

Martin Nečaský

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

{mlynkova,necasky}@ksi.mff.cuni.cz
http://www.ksi.mff.cuni.cz/~mlynkova/
http://www.ksi.mff.cuni.cz/~necasky/

Introduction

- XML = a standard for data representation and manipulation
  - XML documents + XML schema
    - DTD, XML Schema, Schematron, RELAX NG, …
- Why schema?
  - Known structure, valid data, limited complexity ⇒ Optimization
- Problems of real-world data:
  - Users do not use schemas at all
  - Schema = a kind of documentation
  - XML Schema language is not used
- Solution: Automatic inference of XML schema S_D for a given set of documents D

Existing Approaches

- Fact: XML schema = extended context-free grammar

- Classical steps:
  1. Derivation of initial grammar (IG)
     - For each element E and its subelements $E_1, E_2, \ldots, E_n$ we create production $E \rightarrow E_1 E_2 \ldots E_n$
  2. Clustering of rules of IG
  3. Construction of prefix tree automaton (PTA) for each cluster
  4. Generalization of PTAs
     - Merging state algorithms
     - Multiple solutions:
       - We need to evaluate the quality of a solution
       - Too general vs. too restrictive
  5. Expressing the inferred REs in target XML schema language
     - Most common: Direct rewriting of REs to content models

Example (1)

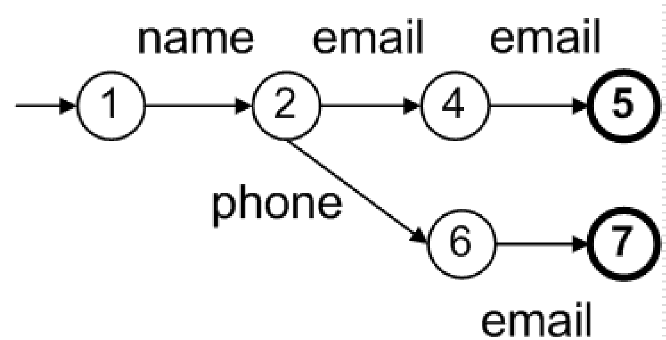
```
...
<person id="123">
  <name>
    <first>Irena</first>
    <surname>Mlynkova</surname>
  </name>
  <email>irena.mlynkova@gmail.com</email>
  <email>irena.mlynkova@mff.cuni.cz</email>
</person>
<person id="456" holiday="yes">
  <name>
    <surname>Necasky</surname>
    <first>Martin</first>
  </name>
  <phone>123-456-789</phone>
  <email>martin.necasky@mff.cuni.cz</email>
</person>
...
```

person → name email email
person → name phone email

name → first surname
name → surname first

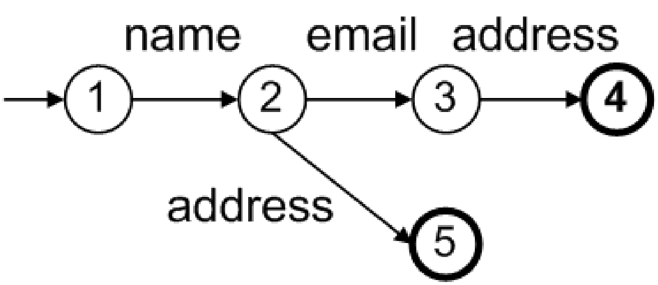
first → PCDATA
surname → PCDATA
email → PCDATA
phone → PCDATA

person:

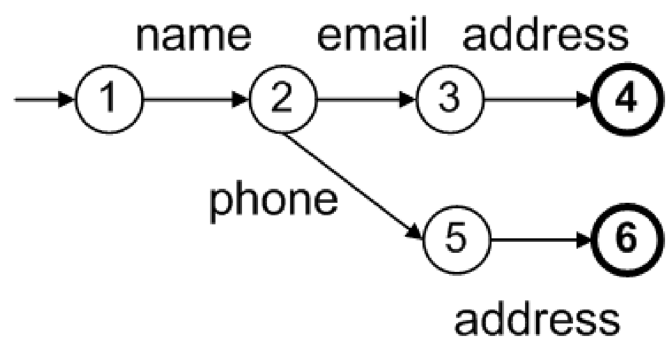


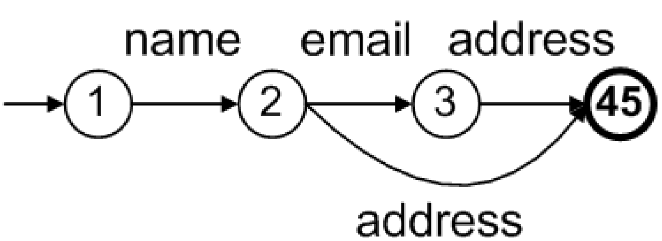
Example (2)

person → name email address
person → name address

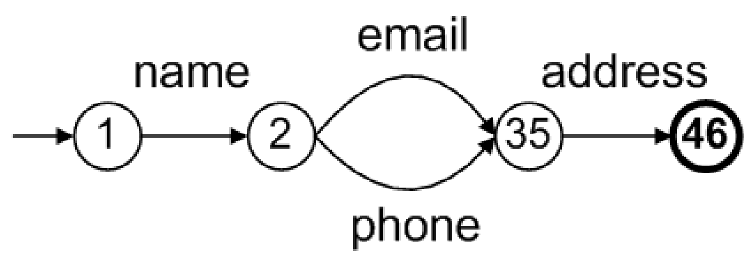


person → name email address
person → name phone address





person → name email? address



person → name (email | phone) address

Our Approach

- Observation 1: XML Schema language involves plenty of structurally equivalent constructs
  - Naïve approach:
    1. Identify sub-schemas with multiple equivalent expressions
    2. Offer the options to a user
  - Problem: too many possibilities
- Observation 2: XML data can bear additional information
- Aim: Reduction of possibilities, exploitation of other information $\Rightarrow$ more realistic schemas
  - Shared fragments
  - Semantics of element/attribute names
  - Data statistics

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

Equivalence Classes

```
<xs:attribute name="holiday">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="yes"/>
      <xs:enumeration value="no"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

```
<xs:attribute name="holiday" type="typeHoliday"/>

<xs:simpleType name="typeHoliday">
  <xs:restriction base="xs:string">
    <xs:enumeration value="yes"/>
    <xs:enumeration value="no"/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:complexType name="typeName">
  <xs:all>
    <xs:element name="first" type="xs:string"/>
    <xs:element name="surname" type="xs:string"/>
  </xs:all>
</xs:complexType>
```

```
<xs:complexType name="typeName">
  <xs:choice>
    <xs:sequence>
      <xs:element name="first" type="xs:string"/>
      <xs:element name="surname" type="xs:string"/>
    </xs:sequence>
    <xs:sequence>
      <xs:element name="surname" type="xs:string"/>
      <xs:element name="first" type="xs:string"/>
    </xs:sequence>
  </xs:choice>
</xs:complexType>
```

Shared Fragments

- Observation: Globally defined schema fragments are usually shared
  - Important information for further processing
- Ex. 1: Element E and F have a common set of attributes $\Rightarrow$ attributeGroup
- Problems:
  - Still too many solutions
    - attribute vs. attributeGroup vs. complexType, …
  - We could merge schema fragments having nothing in common
- Ex. 2:
  - person: id, name, address, phone
  - book: id, name, address, authors

Semantics of Element/Attribute Names

- Observation: We should merge only semantically related fragments
- Ex. 1:
  - Related terms: person, author, editor, manager, …
  - Unrelated terms: person, book, …
- Ex. 2: employee is a broader term to manager

  ⇒ Hierarchy of complex types
- Problems:
  - The common fragments to be shared can be small
    - Merging singletons?
  - Common terms (e.g. id, comment, name, item, …) ⇒ stop list
  - Homonymy ⇒ broader context

Data Statistics

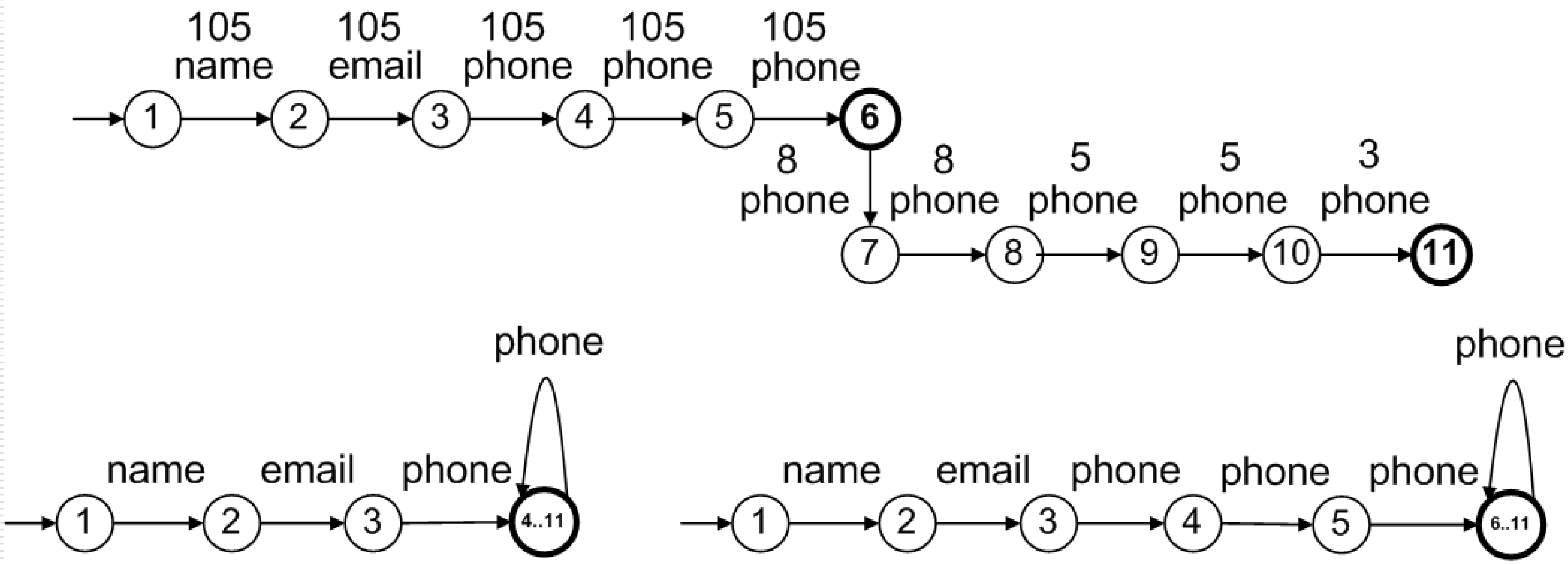
- Observation: The data in D provide additional information
- Existing approaches: focus on concise and precise REs
- Ex. 1:
  - a, a, a, a, a = a+
  - (a, b)|(a, c) = a, (b|c)
- Ex. 2: in 95% of cases person has 2 phone numbers, in 5% of cases more
  - phone+
  - phone phone phone*

Proposed Algorithm

- Modification of a classical merging state algorithm
- We need:
  1. to add new merging rules
     - Splitting repetition
  2. to enable splitting an automaton into multiple ones
     - Outlining a globally defined fragment to be shared
  3. to make automaton modifications with regard to other automata
  4. to modify the evaluating function

Example 1. Splitting Repetitions

	100	100	100	100	100					
person →	name	email	phone	phone	phone					
	5	5	5	5	5	8	8	5	5	3
person →	name	email	phone	phone	phone	phone	phone	phone	phone	phone



person → name email phone+

person → name email phone phone phone phone*

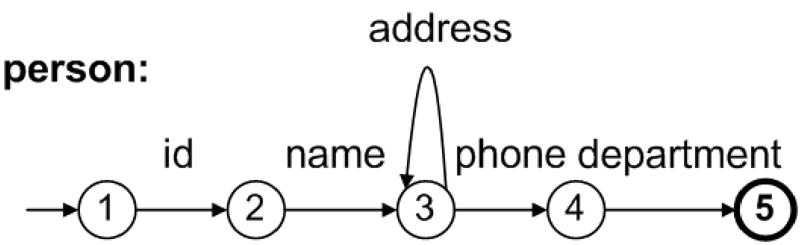
Example 2. Outlining of Schema Fragments

person → id name address* phone department
manager → id name address* phone office

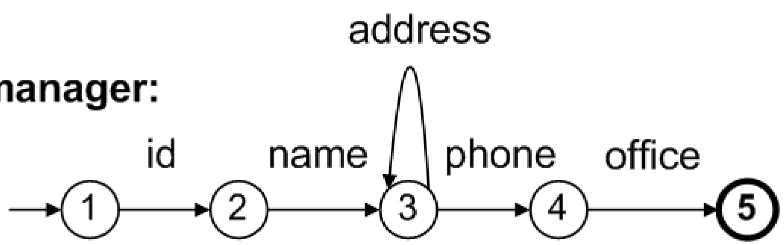
common_items → id name address* phone

person → common_items department
manager → common_items office

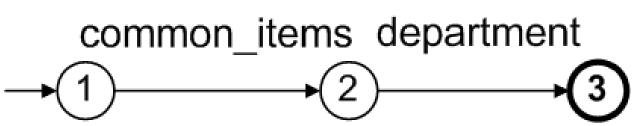
person:



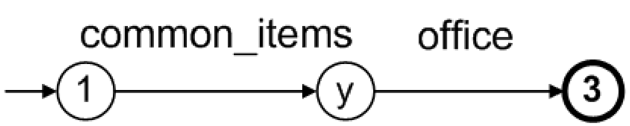
manager:



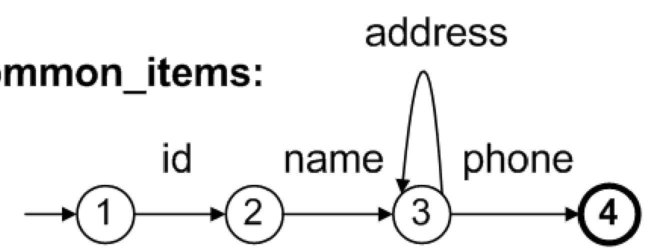
person:



manager:



common_items:



Evaluating Function

- Current approaches: MDL principle
  - Schema should be enough general $\Rightarrow$ low number of states of automata
    - Size in bits of productions in S_D
  - Schema should preserve details $\Rightarrow$ expresses document instances using short codes
    - Size in bits of document instances D expressed using productions in S_D
- Problem: Classical MDL principle would disadvantage splitting repetitions and outlining schema fragments

Outlining

- Problem: Outlining decreases the size of S_D, but increases the size of codes
  - Decreases the length of the original productions
  - Adds a new production
- Solution: weight α expressing appropriateness of outlining a production
  - Similarity, stop list, size, context, …

$$\alpha(\vec{x}) = 1 - \Big(\alpha_1 \times sim_{i=1}^{|context(\vec{x})|}(q_i \in context(\vec{x})) + \alpha_2 \times \frac{|context(\vec{x})|}{|R'_s|} + \alpha_3 \times \frac{|model'(\vec{x})| - |stoplist(model'(\vec{x}))|}{avg_{i=1}^{k}(|model'(\vec{q}_i)|)} \Big)$$

Splitting Repetitions

- Problem: Splitting repetitions increases the size of a schema S_D
  - Increases the length of a production
- Solution: weight β expressing the usage of a rule in instances
  - The more a production is used in the instances, the lower the weight is

$$\beta(\vec{x'}) = 1 - \frac{score(\vec{x'})}{score(\vec{x})}$$

$$score(\vec{x}) = \sum_{k=1}^{l} ind(r_{(k)})$$

$$score(\vec{x'}) = \sum_{k=1}^{i-1} ind(r_{(k)}) + rep_{min} * max_{k=i}^{j}(ind(r_{(k)})) + \sum_{k=j+1}^{l} ind(r_{(k)})$$

Expressing Schema in XSD

- Existing works: Straightforward process (automaton $\Rightarrow$ regular expression)
- Our case: outlined productions $\Rightarrow$ multiple options
- Solution: thesaurus
  - Broader term, related term, narrower term, …

m → a (b | c) u x y*   n → a (b | c) x y*

P → a (b | c)   Q → x y*   m' → P u Q   n' → P Q

```
<xs:group name="P">...

<xs:group name="Q">...

<xs:element name="m">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="P"/>
      <xs:element name="u".../>
      <xs:group ref="Q"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="n">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="P"/>
      <xs:group ref="Q"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

m and n are not related, but have similar content models

```
<xs:complexType name="P">...

<xs:group name="Q">...

<xs:element name="m">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="P">
        <xs:sequence>
          <xs:element name="u" .../>
          <xs:group ref="Q"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="n">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="P">
        <xs:sequence>
          <xs:group ref="Q"/>
        </xs:sequence>
      </xs:extension>
    <xs:complexContent>
  </xs:complexType>
</xs:element>
```

m is related to n (cat and dog)

```
<xs:group name="P">...

<xs:group name="Q">...

<xs:complexType name="mT">
  <xs:sequence>
    <xs:group ref="P"/>
    <xs:element name="u" .../>
    <xs:group ref="Q"/>
  </xs:sequence>
</xs:extension>

<xs:complexType name="nT">
  <xs:complexContent>
    <xs:restriction base="mT">
      <xs:sequence>
        <xs:group ref="P"/>
        <xs:element name="u"
            maxOccurs="0"/>
        <xs:group ref="Q"/>
      </xs:sequence>
    </xs:restriction>
  <xs:complexContent>
</xs:complexType>

<xs:element name="m"
          type="mT"/>

<xs:element name="n"
          type="nT"/>
```

m broader term of n (employee and director)

Conclusion

- Advantages of algorithm:
  - More realistic results
    - Closer to human-written ones
  - More precise information on the input data
- Current and future work
  - Implementation
    - Other improvements $\Rightarrow$ mutual comparison of impact
  - Exploitation
    - Storage strategies of XML data
  - Further improvements
    - User interaction, inference of more precise integrity constraints, other schema languages (RELAX NG, Schematron)…

Thank you