XML Schema Inference: A Study
(Technical Report)

Irena Mlýnková

Charles University
Faculty of Mathematics and Physics
Department of Software Engineering
Malostranske nam. 25
118 00 Prague 1, Czech Republic
Email: irena.mlynkova@mff.cuni.cz

Abstract. The XML has undoubtedly become a standard for data representation and manipulation. But most of XML documents are still created without the respective description of their structure, i.e. an XML schema. Hence, in this paper we focus on the problem of automatic inferring of an XML schema for a given sample set of XML documents. We provide an overview and analysis of existing approaches and compare their key advantages. We conclude the text with a discussion of open issues and problems to be solved as well as their possible solutions.

1 Introduction

Without any doubt the XML [8] is currently a de-facto standard for data representation. Its popularity is given by the fact that it is well-defined, easy-to-use and, at the same time, enough powerful. To enable users to specify own allowed structure of XML documents, so-called *XML schema*, the W3C[1] has proposed two languages – DTD [8] and XML Schema [7, 33]. The former one is directly a part of XML specification and due to its simplicity it is one of the most popular formats for schema specification. The latter language was proposed later, in reaction to the lack of constructs of DTD. The key emphasis is put on simple types, object-oriented features (such as user-defined data types, inheritance, substitutability etc.) and reusability of parts of a schema or whole schemas.

On the other hand, statistical analyses of real-world XML data show that a significant portion of XML documents (52% [20] of randomly crawled or 7.4% [24] of semi-automatically collected[2]) still have no schema at all. What is more, XML Schema definitions (XSDs) are used even less (only for 0.09% [20] of randomly crawled or 38% [24] of semi-automatically collected XML documents) and even if they are used, they often (in 85% of cases [4]) define so-called *local tree grammars* [27], i.e. languages that can be defined using DTD as well.

[1] http://www.w3.org/
[2] Data collected with the interference of a human operator.

In reaction to this situation a new research area of automatic construction of an XML schema has opened. The key aim is to create an XML schema for the given sample set of XML documents that is neither too general, nor too restrictive. It means that the set of document instances of the inferred schema is not too broad in comparison with the provided set of sample data but, also, it is not equivalent to the sample set. Currently, there are several proposals of respective algorithms, but there is still a space for further improvements. In this paper we provide an analysis and overview of existing approaches and compare their advantages and disadvantages. In particular, we deal with the problems that have already been solved and solutions used. We conclude the text with a discussion of remaining open issues as well as their possible solutions.

The paper is structured as follows: Section 2 provides an introduction to existing languages for definition of XML schema and their relation to theory of languages and automata. In Section 3 we discuss the existing algorithms for schema inference and in Section 4 we sum up the key findings. Section 5 discusses the related open issues and Section 6 provides conclusions.

2 Schema Definition Languages

A natural requirement for an XML document is *well-formedness*, i.e. conformance to a set of requirements that ensure correct tree structure. Nevertheless, if we want to share or exchange XML data between multiple users, it is necessary to provide also a schema that describes their allowed structure more precisely.

DTD The simplest and most popular language for description of the allowed structure of XML documents is currently the Document Type Definition (DTD) [8]. It enables one to specify allowed elements, attributes and their mutual relationships, order and number of occurrences of subelements (using operators '`,`', '`|`', '`?`', '`+`' and '`*`'), data types (`ID`, `IDREF`, `IDREFS`, `CDATA` or `PCDATA`) and allowed occurrences of attributes (`IMPLIED`, `REQUIRED` or `FIXED`). A simple example of a database of employees is depicted in Figure 1.

At first glance it seems that the specification of the allowed structure is sufficient. Nevertheless, even in this simple example we can find several problems. For instance, we are not able to specify the correct structure of an e-mail address. Similarly, we cannot simply specify that a person can have four e-mail addresses at maximum. And, as we can see, the fact that the order of elements `first` and `surname` is not significant cannot be expressed simply as well. Therefore, the W3C proposed a more powerful tool – the XML Schema language.

XML Schema The XML Schema language [7,33] has a number of advantages as well as disadvantages. The main advantages are that:

- each XSD is a well-formed XML document,
- it has a strong support of data types, both simple and complex and both built-in and user-defined,

```
<!ELEMENT employees (person)+>
<!ELEMENT person (name, email*, relationships?)>
  <!ATTLIST person id ID #REQUIRED>
  <!ATTLIST person note CDATA #IMPLIED>
  <!ATTLIST person holiday (yes|no) "no">
<!ELEMENT name ((first, surname)|(surname, first))>
<!ELEMENT first (#PCDATA)>
<!ELEMENT surname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT relationships EMPTY>
  <!ATTLIST relationships superior IDREF #IMPLIED
                          inferior IDREFS #IMPLIED>
```

Fig. 1. An example of a DTD of employees

- it enables one to re-use and re-define existing schemas or their selected parts,
- it enables one to specify the allowed structure using more precise constraints (e.g. minimum and maximum allowed occurrences, ordered/unordered sequences, integrity constraints etc.) and
- it enables one to specify equivalent schemas using distinct constructs.

For example an XSD equivalent to the example of a DTD in Figure 1 is depicted in Figure 2.

Note that XML Schema has also several disadvantages. In particular, as we can from the example, the syntax of the language is more complex and more space-consuming than in case of DTDs. And the second disadvantage is the same as the last mentioned advantage. It highly complicates the automatic processing of XSDs, since all the possibilities must be taken into account.

In general, the constructs of XML Schema can be divided into *basic*, *advanced* and *auxiliary*. The basic constructs involve simple data types (`simpleType`), complex data types (`complexType`), elements (`element`), attributes (`attribute`), groups of elements (`group`) and groups of attributes (`attributeGroup`). Simple data types involve both built-in data types (except for `ID`, `IDREF`, `IDREFS`), such as, e.g., `string`, `integer`, `date` etc., as well as user-defined data types derived from existing simple types using `simpleType` construct. Complex data types enable one to specify both content models of elements and their sets of attributes. The content models can involve ordered sequences (`sequence`), choices (`choice`), unordered sequences (`all`), groups of elements (`group`) or their allowable combinations. Similarly, they enable one to derive new complex types from existing simple (`simpleContent`) or complex types (`complexContent`). Elements simply join simple/complex types with respective element names and, similarly, attributes join simple types with attribute names. And, finally, groups of elements and attributes enable one to globally mark selected schema fragments and exploit them repeatedly in various parts using so-called *references*. In general, basic constructs are present in almost all XSDs.

The set of *advanced* constructs involves type substitutability and substitution groups, identity constraints (`unique`, `key`, `keyref`) as well as related simple

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="employees">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="person" minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="person">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="name"/>
    <xs:element name="email" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="relationships" minOccurs="0" maxOccurs="1"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID" use="required"/>
   <xs:attribute name="note" type="xs:string"/>
   <xs:attribute name="holiday" default="no">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="yes"/>
      <xs:enumeration value="no"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:complexType>
 </xs:element>

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

 <xs:element name="relationships">
  <xs:complexType>
   <xs:attribute name="superior" type="xs:IDREF"/>
   <xs:attribute name="inferior" type="xs:IDREFS"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Fig. 2. An example of an XSD of employees

data types (`ID`, `IDREF`, `IDREFS`) and assertions (`assert`, `report`). Type substitutability and substitution groups enable one to change data types or allowed location of elements. Identity constraints enable one to restrict allowed values of elemets/attributes to unique/key values within a specified area and to specify references to them. Similarly, assertions specify additional conditions that the values of elements/attributes need to satisfy, i.e. they can be considered as an extension of simple types.

The set of *auxiliary* constructs involves wildcards (`any`, `anyAttribute`), external schemas (`include`, `import`, `redefine`), notations (`notation`) and annotations (`annotation`). Wildcards and external schemas combine data from various XML schemas. Notations bear additional information for superior applications. And annotations can be considered as a kind of advanced comments.

2.1 Relation to Automata and Grammars

An XML schema describing the allowed structure of XML documents is a context-free grammar [3], i.e. a grammar where nonterminals can be rewritten without regard to the context in which they occur.

Definition 1. A context-free grammar *is quadruple* $G = (N, T, P, S)$*, where* N *and* T *are finite sets of* nonterminals *and* terminals*,* P *is a finite set of* productions *and* S *is a non terminal called a* start symbol*. Each production is of the form* $A \rightarrow \alpha$*, where* $A \in N$ *and* α *is a regular expression over alphabet* $(N \cup T)^*$*.*

The language generated by grammar G *is denoted by* $L(G)$*.*

Definition 2. *Given the alphabet* Σ*,* a regular expression *(RE) over* Σ *is inductively defined as follows:*

- $\emptyset$ (empty set) *and* ϵ (empty string) *are REs.*
- $\forall a \in \Sigma$ *: a is a RE.*
- *If* r *and* s *are REs of* Σ*, then* (rs) (concatenation)*,* $(r|s)$ (alternation) *and* (r^*) Kleene closure) *are REs.*

Note that the DTD language adds two abbreviations: $(s|\epsilon) = (s?)$ and $(ss^*) = (s^+)$. Also the concatenation is expressed via the '`,`' operator. The XML Schema language adds (among other extensions) another one, so-called *unordered sequence* of REs $s_1, s_2, ..., s_k$, i.e. an alternation of all possible ordered sequences of $s_1, s_2, ..., s_k$.

A language specified by a grammar can be accepted by an automaton, in our case a finite state automaton.

Definition 3. A finite state automaton (FSA) *is quintuple* $A = (Q, \Sigma, \delta, S, F)$*, where* Q *is a set of states,* Σ *is a set of input symbols (alphabet),* $\delta : Q \times \Sigma^* \rightarrow Q$ *is the transition function,* $S \in Q$ *is the start state and* $F \subseteq Q$ *is the set of final states.*

The language recognized by an automaton A *is denoted by* $L(A)$*.*

Note that for each RE we can construct a FSA and vice versa.

3 Analysis of Existing Approaches

The studied problem can be described as follows: Being given a set of XML documents $\Delta = \{D_1, D_2, ..., D_n\}$ (i.e. words over an alphabet T), we search for an XML schema S_Δ (i.e. a grammar G_Δ) s.t. $\forall i \in [1, n] : D_i$ is valid against S_Δ (i.e. $\Delta \subseteq L(G_\Delta)$). In particular, we are searching for S_Δ that is enough concise, precise and, at the same time, general.

The existing solutions to the problem of automatic construction of an XML schema can be classified according to several criteria. Probably the most interesting one is the type of the result and the way it is constructed.

From the point of view of the result, we can distinguish methods which output DTDs or XSDs. The problem is that some of the methods claim to produce XSDs, but their expressive power is not beyond the expressive power of DTD. Hence we distinguish only true XSD producing methods. Since most of the DTD constructs are intended for specification of content models of elements, the existing approaches focus mainly on them. Consequently, they often ignore attributes, mixed content or special data types, such as `ID`, `IDREF(S)`. In case of XSD constructs the existing methods focus on simple data types, elements having various contexts and "syntactic sugar" such as unordered sequences.

On the other hand, the types of the inference process can be divided into *heuristic* and *grammar-inferring*. In the former case the result does not belong to any class of grammar and, hence, we cannot say anything about its features. In the latter case the algorithms output particular class of languages with specific characteristics. Although grammars accepting XML documents are context-free, the problem can be reduced to inferring of a set of REs, each for a single element. But, since according to Gold's theorem [14] regular languages are not identifiable only from positive examples (i.e. sample XML documents which should conform to the resulting schema), the existing methods need to exploit either heuristics or restriction to an *identifiable* subclass of regular languages.

Most of the existing approaches use the following strategy: For each occurrence of element $e \in \Delta$ and its subelements $e_1, e_2, ..., e_k$ we construct a production $e \rightarrow e_1\ e_2\ ...\ e_k$. The left hand side is called *element type*, the right hand side is called a *content model* of the element type. The productions form so-called *initial grammar*. For each element type the productions are then merged, simplified and generalized using various methods and criteria. A common approach is so-called *merging state algorithm*, where a *prefix tree automaton* is built from the productions of the same element type and the automaton is generalized via merging of its states. Finally, the generalized grammar/automaton is expressed in syntax of the respective XML schema language.

An example of an initial grammar and prefix tree automaton is depicted in Figure 3.

3.1 Heuristic Approaches

Heuristic approaches are based on experience with manual construction of schemas. Their results do not belong to any special class of grammars and they are based

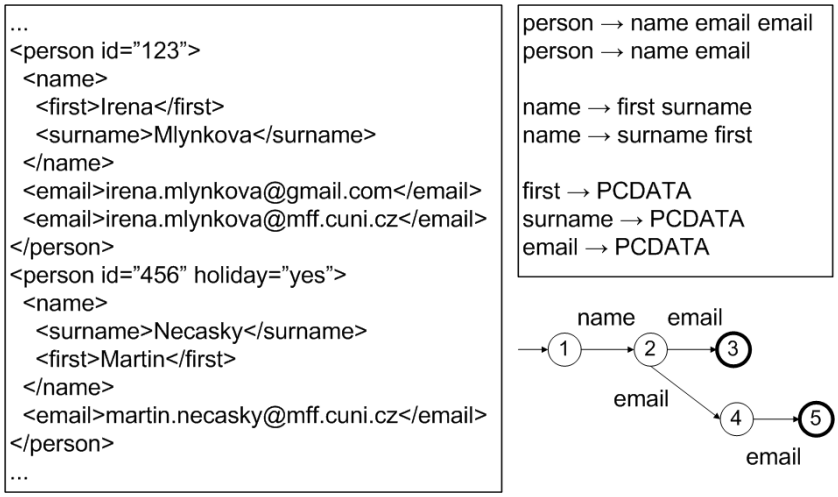

Fig. 3. An example of an initial grammar and a prefix tree automaton for element `person`

on generalization of the initial grammar using a set of predefined heuristic rules, such as, e.g., "if there are more than three occurrences of an element, it is probable that it can occur arbitrary times". These techniques can be further divided into methods which generalize the initial grammar until a satisfactory solution is reached (e.g. [25, 30]) and methods which generate a number of candidates and then choose the optimal one (e.g. [13]). While in the first case the methods are threatened by a wrong step which can cause generation of a suboptimal schema, in the latter case they have to cope with space overhead and specifying a reasonable function for evaluation of quality of the candidates.

GB-engine Probably the first true XML approach to automatic DTD inference is system called *grammar builder engine* (*GB-engine*) [30]. In this particular case it outputs SGML DTDs – a predecessor of XML DTDs. The approach is relative simple and straightforward. Firstly, the initial grammar is created. Then the productions with the same element type are combined using the alternation into a common production:

```
A -> B
A -> C            => A -> B | C | D E
A -> D E
```

In the next step, using a set of heuristic rules the content models of the productions are simplified and generalized until any of the rules can be applied. Apart from simple rules that solve trivial cases, the set involves rules dealing with repetitions, identical bases, redundancy etc.:

```
A -> B B B B        => A -> B+
```

```
A -> (B C)* | B? C? => A -> (B? C?)*
A -> B C | B C*     => A -> B C*
```

Finally, the productions are rewritten into DTD syntax. Note that apart from the content models, the system supports attributes, `#PCDATA` and mixed content.

DTD-miner *DTD-miner* [25] is one of the first approaches inferring an XML DTD. It is very similar to the previous case, the key difference is in the representation of the input XML documents and in the heuristic rules. Firstly, the so-called *spanning graph*, i.e. an equivalent of well-known *data guide* [15], is built for the input XML documents. Each node of the graph represents a unique element $e \in \Delta$ and bears information on all its attributes and textual data. The edges of the graph represent element-subelement relationships occurring in Δ and their occurrence.

The set of rules involves optionality, repetition and grouping. Optionality identifies elements that do not occur in all the input documents. Repetition identifies adjacent elements that occur multiple times. And grouping rule identifies repeating groups of adjacent elements. Finally, the generated DTD is refined to gain less complex structures using further rules such as:

```
A?, B?, C? => A | B | C
A, B, C, D, A, D, B, C => (A | B | C | D)+
```

Note that while in the previous case the alternation was the first to be involved in the result, in this case it is the last one.

XTRACT The *XTRACT* [13] system is a classical representative of a merging state algorithm. It differs in two aspects: The approach produces a set of possible solutions and selects the optimal one, i.e. it is able to evaluate quality of a schema generalization. The possible solutions are created using heuristic generalization rules for optionality, repetition and grouping similar to those proposed in DTD-miner.

For the purpose of schema evaluation the authors exploit so-called *minimum description length (MDL) principle*. It expresses the quality of a DTD candidate using two aspects – conciseness and preciseness. *Conciseness* of a DTD is expressed using the number of bits required to describe the DTD (the smaller, the better). *Preciseness* of a DTD is expressed using the number of bits required for description of the input data using the DTD. In other words, the more accurately the structure is described, the fewer bits are required. Since the two conditions are contradictory, their balancing brings reasonable and realistic results.

sk-ANT The *sk-ANT* [35] method extends the previous approach in two aspects. Firstly, the searching for the optimal solution is performed using the *Ant Colony Optimization (ACO)* heuristics and a new merging method called *sk-strings* is introduced.

The ACO heuristics is a kind of general heuristics that enables one to find a suboptimal solution. A set of artificial "ants" $B = \{a_1, a_2, ..., a_l\}$ search a space

S of possible solutions (i.e. DTD generalizations) trying to find the optimal solution. The quality of a solution is evaluated using the MDL principle. In i-th iteration each $a \in B$ searches a subspace of S for a local suboptimum until it "dies" after performing a predefined amount of steps. A step of an ant represents an application of any of the merging criterions on the current DTD. While searching, an ant a spreads a certain amount of "pheromone", i.e. a positive feedback which denotes how good solution it has found so far. This information is exploited by ants from the following iterations to choose better search steps.

On the other hand, the sk-strings merging rule is based on a relaxed variant of Nerode equivalence. *Nerode equivalence* assumes that two states p and q are equivalent if sets of all paths leading from p and q to terminal state(s) in F are equivalent. But as such condition is hardly checked, we can restrain to k-strings, i.e. only paths of length of k or paths terminating in a terminal state. The respective equivalence of states then depends on equivalence of sets of outgoing k-strings. In addition, for easier processing we can consider only s most probable paths, i.e. we can ignore singular special cases.

While the ACO heuristics enables one to avoid constructing multiple solutions concurrently, the sk-string provides a better merging criterion.

ECFG The so far described approaches focus on inferring DTDs, in particular respective content models. But, since the XML Schema language offers wider range of constructs, there also appear heuristic approaches dealing with pure XSD structures. Probably the first representative is proposed in paper [9].

The approach focuses on inference of content models consisting of complex types, sequences and choices, simple data types and exact occurrence ranges. Using an XSD formalism – so-called *extended context-free grammars (ECFG)* – the authors extend a classical merging state algorithm with preserving the exact ranges of occurrences and adding a step which infers simple data types. For this purpose each set of values of an element/attribute is analyzed to identify the minimal data type which contains all of them. Nevertheless, the authors focus only on numeric data types (such as `decimal`, `float`, `long`, `negativeInteger`), `date`, `binary` and `string`.

SchemaMiner The *SchemaMiner* system proposed in [34] is another representative of an inference approach that deals with true XSD constructs. It focusses on inferring elements with the same name but different structure and unordered sequences. For this purpose the authors exploit ideas from the previously described works, such as ACO heuristics, sk-strings, (k, h)-context [2] or MDL principle in combination with exploitation of tree and graph similarity and clustering.

The elements with the same name but different content are supported only in XSDs and their inference requires exploitation of more sophisticated approaches than combining productions with the same element type. On the other hand, although the unordered sequences are a classical example of XSD "syntactic sugar", their exploitation enables to define simple and, hence, realistic and usable schemas.

3.2 Inference of a Grammar

Methods *inferring a grammar* exploit the theory of languages and grammars and thus ensure a certain degree of quality of the result. They are based on the idea that we can view an XML schema as a grammar and an XML document valid against the schema as a word generated by the grammar. As we have mentioned, since the class of the regular languages is not identifiable from positive examples, the grammar-inferring methods focus on its identifiable subclasses. All the approaches are classical merging state algorithms, whereas the merging criteria are mostly directly defined by the characteristics of the output class of the language.

(k, h)-contextual languages Paper [2] is probably the first approach dealing with inference of a particular class of XML languages. The approach is based on the observation that if a sufficiently long sequence of terminals occurs in two places in the examples, the components that follow are independent on the position of the sequence within the document.

Definition 4. *A regular language L is* k-contextual*, if there exists a finite automaton A s.t. $L = L(A)$ and for any two states p_k, q_k of A and all input symbols $a_1a_2...a_k$: if there are two states p_0, q_0 of A s.t. $\delta(p_0, a_1a_2...a_k) = p_k$ and $\delta(q_0, a_1a_2...a_k) = q_k$, then $p_k = q_k$.*

Definition 5. *A regular language L is* (k,h)-contextual*, if there exists a finite automaton A s.t. $L = L(A)$ and for any two states p_k, q_k of A and all input symbols $a_1a_2...a_k$: if there are two states p_0, q_0 of A s.t. $\delta(p_0, a_1) = p_1$, $\delta(p_1, a_2) = p_2$, ..., $\delta(p_{k-1}, a_k) = p_k$ and $\delta(q_0, a_1) = q_1$, $\delta(q_1, a_2) = q_2$, ..., $\delta(q_{k-1}, a_k) = q_k$, then $p_i = q_i$ for every i s.t. $0 \leqslant h \leqslant i \leqslant k$.*

The k-contextual and (k, h)-contextual languages form two identifiable subclasses of regular languages which assume that the context of elements is limited. The algorithm is a classical merging state approach starting with a prefix tree automaton, but the merging is not made on the basis of heuristics, but on the basis of the respective features of the languages. The merging criterion is based on an assumption that two states p_k and q_k of the automaton are identical (i.e. can be merged) if there exist identical paths of length k terminating in p_k and q_k. In case of (k, h)-context, also h preceding states in these paths are then identical. The resulting grammar is finally refined to acquire more realistic and concise result.

f-distinguishable languages A different class of identifiable regular languages is inferred in [12]. These are so-called *f-distinguishable languages*.

Definition 6. *Let T be a set of terminals and F some finite set. A mapping $f : T^* \to F$ is called* a distinguishing function*, if $f(w) = f(z)$ implies $f(wu) = f(zu)$ for all $u, w, z \in T^*$.*

Language $L \in T^*$ is called f-distinguishable if, for all $u, v, w, z \in T^*$ with $f(w) = f(z)$, we have $zu \in L \Leftrightarrow zv \in L$ whenever $\{wu, wv\} \in L$.

Being given a set of positive examples Δ and the distinguishing function f, the authors propose a merging state algorithm that constructs an automaton A accepting that smallest f-distinguishable language that contains Δ.

1-unambiguity An important aspect of XML schemas is so-called *1-unambiguity*. According to the W3C specification, all content models in an XML schema must be 1-unambiguous (deterministic), i.e. they can be matched without looking ahead. A simple example of an ambiguous (non-deterministic) content model is $(e_1, e_2)|(e_1, e_3)$, where while reading e_1 we are not able to decide which of the alternatives to choose unless we read the following element. Though this topic is for some research groups controversial and there exist several studies dealing with (un)necessity of this constraint [18], this condition still remains valid. On the other hand, we can find XML parsers and validators that are able to process also ambiguous content models.

Probably for the first time this problem has been faced in paper [21]. For the purpose of preserving the 1-unambiguity, the authors restrict to so-called *single-occurrence* property of all derived content models which ensures the 1-unambiguity.

Definition 7. A single-occurrence regular expression (SORE) *is a regular expression α over Σ s.t. each $s \in \Sigma$ occurs at most once in α.*

The authors propose a set of heuristic transformation rules that modify and generalize the initial grammar so that the single-occurrence property is fulfilled in the result. An extension of the proposed approach for XSDs involving simple data types and attributes (which are not supported in the original method) has recently been implemented in system *XStruct* [16].

SOREs and CHAREs The strategy of paper [5] – to define an identifiable class of regular languages and respective inference algorithm – is similar to the previous ones, but the motivation is slightly different. The authors result from their analysis of real-world XML data and XML schemas and define the classes so that they cover most of the real-world examples. Hence, contrary to the previous works based purely on results of theory of languages, the usability of this approach is undeniable.

The authors focus on two classes of identifiable REs to be inferred – the previously defined SOREs and new, so-called *chain regular expressions (CHAREs)*.

Definition 8. A chain regular expression (CHARE) *over Σ is a SORE over Σ that consists of a sequence of factors $f_1 f_2 ... f_n$, where every factor is an expression of the form $(a_1|a_2|...|a_k)$, $(a_1|a_2|...|a_k)?$, $(a_1|a_2|...|a_k)^+$ or $(a_1|a_2|...|a_k)^*$, where $k \geqslant 1$ and every $a_i \in \Sigma$.*

The motivation for focusing on CHAREs results from authors' experience with inferring DTDs for real-world XML data. They discover that for small data sets the SOREs are too rich and inference of CHAREs provides more realistic and concise results. Similarly to the previous cases, both the algorithms are based on merging states of a prefix tree automaton using rules that ensure that the result belongs to the required class.

k-local single-occurrence grammars Following their previous work [5], the authors have recently focussed on features and properties of real-world XSDs [6]. Using a similar strategy, they first discover a subclass of XSDs that is most common in real-world XML data (occurs in 98% cases) and, at the same time, that can be identified only from positive examples – so called *k-local, single-occurrence XSDs*.

Definition 9. *An XSD is* k-local*, if its content models depend only on labels up to the* k*-th ancestor.*

The authors then propose a theoretically complete merging state algorithm called *i*XSD that enables one to infer k-local, single-occurrence XSDs.

4 Summary

The key characteristics of the described approaches are summed up in Tables 1 and 2.

Name	**Schema**	**Key Advantages**
GB-engine [30]	SGML DTD	First simple heuristic rules.
DTD-miner [25]	DTD	Spanning graph, heuristic rules for optionality, repetition and grouping.
XTRACT [13]	DTD	Set of candidate solutions, MDL principle.
sk-ANT [35]	DTD	*sk*-string merging (based on Nerode equivalence), ACO heuristics.
ECFG [9]	XSD	Precise occurrence ranges, simple data types.
SchemaMiner [34]	XSD	Unordered sequences, elements with the same name, but different structure.

Table 1. Key characteristics of heuristic methods

Name	**Schema**	**Key Advantages**
[2]	DTD	k-contextual and (k, h)-contextual languages.
[12]	DTD	f-distinguishable languages.
[21]	DTD	1-unambiguity.
[5]	DTD	Single-occurrence and chain REs, based on knowledge of real-wold data.
[6]	XSD	k-local single-occurrence XSDs, based on knowledge of real-wold data.

Table 2. Key characteristics of grammar-inferring methods

5 Open Issues

Although each of the existing approaches brings certain interesting ideas and optimizations, there is still a space of possible future improvements. We describe and discuss them in this section.

User Interaction In all the existing papers the approaches focus on automatic inference of an XML schema. The problem is that the resulting schema may be highly unnatural. Although e.g. the MDL principle evaluates the quality of the schema using a realistic assumption that it should tightly represent the data and, at the same time, be concise and compact, users' preferences can be quite different. (Note that this is not the same motivation as in case of papers [5, 6] that focus on real-world DTDs and XSDs.) Hence, a natural improvement may be exploitation of user interaction.

For instance, the user may influence the process of merging by proposing preferred merging operations/target constructs, clustering similar elements etc. Such approach will not only enable to find more concise result, but to find it more efficiently as well. Some of the existing papers (e.g. [2]) mention the aspect of user interaction, typically in the final step of refinement of the result, but there seems to be no detailed study and, in particular, respective implementation. And, naturally, this problem is closely related to a suitable user interface which does not require complex operations and decisions.

Other Input Information In all the existing works the XML schema is inferred on the basis of a set of positive examples, i.e. XML documents that should conform to the inferred schema. As we have mentioned, the Gold's theorem highly restricts the existing solutions and, hence, the authors focus on heuristic approaches or limit the methods to particular identifiable classes of languages. But another natural solution to the problem is to exploit additional information, such as XML schema or XML queries.

In the former case we can find the motivation in typical situation [24] when a user creates an XML schema of XML documents but then modifies and updates only the data, whereas the schema is considered as a kind of documentation. Consequently, the schema does not describe the current structure of the data anymore, however it can be used as a source of information because certain matching can be still found. Note that a similar problem is being currently solved in the area of *schema evolution* (e.g. [19]).

In case of exploitation of XML queries the motivation is similar though more obvious. In general, the queries restrict only parts of the data structure (those that should appear on output), however even this partial information can be exploited for schema inference. Similarly to the previous case, a related problem is being solved in the area of *XML views* (e.g. [29]).

In addition, there seems to be no approach that would exploit negative examples (i.e. XML documents that should not conform to the schema). In this case we can find a real-world motivation again in the area of data evolution and versioning.

XML Schema Simple Data Types One of the biggest advantages of the XML Schema language in comparison to DTD is its wide support of simple data types [7]. It involves 44 built-in data types such as, e.g., `string`, `integer`, `date` etc., as well as user-defined data types derived from existing simple types using `simpleType` construct. In enables one to derive new data types using *restriction* of values of an existing type (e.g. a string value having length greater than two), *list* of values of an existing type (e.g. list of integer values) or *union* of values of existing data types (e.g. union of positive and negative integers).

Hence, a natural improvement of the existing approaches is a precise inference of simple data types. Most of the existing approaches omit the simple data types and consider all the values as strings. Two exceptions are proposed in [9,16], but both the algorithms focus only on selected built-in data types.

Note that the necessity to infer simple data types is naturally closely related to the purpose the schema is inferred for. Assuming that the resulting XML schema is used within a kind of XML data editor, the inferring module should propose also simple data types. On the other hand, if the inferred XML schema is used as a solution for approaches based on existence of an XML schema, e.g. schema-driven XML-to-relational mapping methods (e.g. [22, 31]), the simple data types are of marginal importance and, thus, can be omitted.

XML Schema Advanced Constructs The second big advantage of the XML Schema language are various complex constructs. The language exploits object-oriented features, such as user-defined data types, inheritance, polymorphism, i.e. substitutability of both data types and elements etc. Although most of these constructs do not extend the expressive power of XML Schema in comparison to DTD (i.e. they are a kind of "syntactic sugar") [23], they enable one to specify more user-friendly and, hence, realistic schemas. Naturally, their usage is closely related to the previously described problem of user-interaction, since only the user can specify which of the constructs are preferred.

Integrity Constraints Both DTD and XML Schema enable one to specify not only the structure of the data, but also various semantic constraints. Both involve `ID` and `IDREF(S)` data types that specify unique identifiers and references to them. The XML Schema language extends this feature using `unique`, `key` and `keyref` constructs that have the same purpose but enable one to specify the unique/key values more precisely, i.e. for selected subsets of elements and/or attributes and valid within a specified area. In addition, the `assert` and `report` constructs enable one to express specific constraints on values using the XPath language [10]. Unfortunately, none of the existing approaches focusses on any of these constraints. In addition, there are also more general integrity constraints [28] that could be inferred, though they cannot be expressed in the existing schema specification languages so far. In general, their inference would extend the optimization of approaches that analyze and exploit information on XML data from XML schemas.

Currently there exist several works which focus on constraint inference [11, 32], but they focus on restricted cases of integrity constraints in special situ-

ations. There seems to exist no method that would combine schema inference with a more general approach to inference of related integrity constraints.

Other Schema Definition Languages The DTD and XML Schema are naturally not the only languages for definition of structure of XML data, though they are undoubtedly the most popular ones. The obvious reason is that these two have been proposed by the W3C, whereas DTD is even a part of specification of XML. Nevertheless, there are also other relatively popular schema specification languages. The most popular ones are *RELAX NG* [26] and *Schematron* [17] which both have already become ISO standards.

The Relax NG has similar strategy as both XML Schema and DTD, i.e. it describes the structure of XML documents using content models. Contrary to XML Schema it has much simple syntax, whereas contrary to DTD is supports a richer set of simple data types. On the other hand, the Schematron uses completely different strategy. It does not specify a grammar the XML documents should conform to, but a set of conditions, i.e. integrity constraints, the documents must follow. The conditions are expressed using XPath. Hence, while the inference of Relax NG schema can be based on inference of a DTD/XSD without radical modifications, the approach to inference of Schematron constraints will probably require a brand new method. On the other hand, it can be a natural first step towards inferrence of general integrity constraints as described before.

Data Streams A special type of XML data that have only recently become popular and, hence, the necessity for proposing respective processing approaches is crucial are so-called *XML data streams*. In this particular application the input data are so huge that they cannot be kept in a memory concurrently, they cannot be read more than once or their processing cannot "wait" for the last portion of the data. Hence the situation is much more complicated. All the XML technologies are currently being accommodated to stream processing and it is only a matter of time when respective efficient schema inference approach will be required as well.

6 Conclusion

The XML schema of XML documents is currently exploited mainly for two purposes – for data-exchange approaches as a description of the structure and for optimization of various XML-related technologies. In the former case we usually need the inferred schema as a candidate schema further improved by a user using an appropriate editor. In the latter case the approaches exploit the knowledge of the schema for optimization purposes such as finding the optimal storage strategy [22,31] or improving the compression ratio [1]. In general, almost any approach that deals with XML data can benefit from the knowledge of their structure, i.e. XML schema. The only question is to what extent. And the first step towards it are realistic and robust schema inference approaches.

The aim of this paper was to provide an analytical study of existing approaches to XML schema inference, as well as a discussion of remaining open

issues. We have showed that the basic aspects of the problem (such as inference of REs) have successfully been solved. However, there still remain open issues and unsolved problems to focus on. Among others we emphasize especially inference of XML Schema constructs and integrity constraints.

This text should serve as a good starting point for readers searching for an existing solution to their inference problem, as well as those searching for an interesting and practical research topic.

Acknowledgement

This work was supported in part by the National Programme of Research (Information Society Project 1ET100300419).

References

1. *XML-Xpress: High-Performance Schema-Specific Compression for XML Data Formats*. ICT – Intelligent Compression Technologies, Inc., 2000. `http://www.ictcompress.com/products_xmlxpress.html`.
2. H. Ahonen. *Generating Grammars for Structured Documents Using Grammatical Inference Methods*. Report A-1996-4, Dept. of Computer Science, University of Helsinki, 1996.
3. J. Berstel and L. Boasson. XML Grammars. In *Mathematical Foundations of Computer Science*, LNCS, pages 182–191. Springer, 2000.
4. G. J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: a Practical Study. In *WebDB'04: Proc. of the 7th Int. Workshop on the Web and Databases*, pages 79–84, New York, NY, USA, 2004. ACM.
5. G. J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of Concise DTDs from XML Data. In *VLDB'06: Proc. of the 32nd Int. Conf. on Very large data bases*, pages 115–126. VLDB Endowment, 2006.
6. G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML Schema Definitions from XML Data. In *VLDB'07: Proc. of the 33rd Int. Conf. on Very Large Data Bases*, pages 998–1009, Vienna, Austria, 2007. ACM.
7. P. V. Biron and A. Malhotra. *XML Schema Part 2: Datatypes (Second Edition)*. W3C, 2004. `http://www.w3.org/TR/xmlschema-2/`.
8. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C, 2006.
9. B. Chidlovskii. Schema Extraction from XML Collections. In *JCDL'02: Proc. of the 2nd ACM/IEEE-CS Joint Conf. on Digital libraries*, pages 291–292, New York, NY, USA, 2002. ACM.
10. J. Clark and S. DeRose. *XML Path Language (XPath) Version 1.0*. W3C, November 1999. `http://www.w3.org/TR/xpath`.
11. F. Fassetti and B. Fazzinga. FOX: Inference of Approximate Functional Dependencies from XML Data. In *DEXA'07: Proc. of the 18th Int. Conf. on Database and Expert Systems Applications*, pages 10–14, Washington, DC, USA, 2007. IEEE.
12. H. Fernau. Learning XML Grammars. In *MLDM'01: Proc. of the 2nd Int. Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 73–87, London, UK, 2001. Springer.

13. M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: a System for Extracting Document Type Descriptors from XML Documents. In *SIGMOD'00: Proc. of the 2000 ACM SIGMOD Int. Conf. on Management of Data*, pages 165–176, New York, NY, USA, 2000. ACM.
14. E. M. Gold. Language Identification in the Limit. *Information and Control*, 10(5):447–474, 1967.
15. R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *VLDB'97: Proc. of the 23rd Int. Conf. on Very Large Data Bases*, pages 436–445, San Francisco, CA, USA, 1997. Morgan Kaufmann.
16. J. Hegewald, F. Naumann, and M. Weis. XStruct: Efficient Schema Extraction from Multiple and Large XML Documents. In *Proc. of the 22nd Int. Conf. on Data Engineering, Workshops*, page 81, Atlanta, GA, USA, 2006. IEEE.
17. R. Jelliffe. *The Schematron – An XML Structure Validation Language using Patterns in Trees.* 2001. `http://xml.ascc.net/resource/schematron/`.
18. M. Mani. *Keeping Chess Alive: Do We Need 1-Unambiguous Content Models?* talk given at Extreme Markup Languages, Montreal, Canada, 2001.
19. M. Mesiti, R. Celle, M. A. Sorrenti, and G. Guerrini. X-Evolution: A System for XML Schema Evolution and Document Adaptation. In *EDBT'06: Proc. of the 10th Int. Conf. on Extending Database Technology*, LNCS, pages 1143–1146. Springer, 2006.
20. L. Mignet, D. Barbosa, and P. Veltri. The XML Web: a First Study. In *WWW'03: Proc. of the 12th Int. Conf. on World Wide Web, Volume 2*, pages 500–510, New York, NY, USA, 2003. ACM.
21. J.-K. Min, J.-Y. Ahn, and C.-W. Chung. Efficient extraction of schemas for XML documents. *Inf. Process. Lett.*, 85(1):7–12, 2003.
22. I. Mlynkova. A Journey towards More Efficient Processing of XML Data in (O)RDBMS. In *CIT'07: Proc. of the 7th IEEE Int. Conf. on Computer and Information Technology*, pages 23–28, Fukushima, Japan, 2007. IEEE.
23. I. Mlynkova. Similarity of XML Schema Definitions. In *DocEng'08: Proc. of the 8th ACM Symposium on Document Engineering*, Sao Paulo, Brazil, 2008. ACM.
24. I. Mlynkova, K. Toman, and J. Pokorny. Statistical Analysis of Real XML Data Collections. In *COMAD'06: Proc. of the 13th Int. Conf. on Management of Data*, pages 20–31, New Delhi, India, 2006. Tata McGraw-Hill.
25. C.-H. Moh, E.-P. Lim, and W.-K. Ng. Re-engineering Structures from Web Documents. In *DL'00: Proc. of the 5th ACM Conf. on Digital Libraries*, pages 67–76, New York, NY, USA, 2000. ACM.
26. M. Murata. *RELAX (Regular Language Description for XML).* 2002. `http://www.xml.gr.jp/relax/`.
27. M. Murata, D. Lee, and M. Mani. Taxonomy of XML Schema Languages Using Formal Language Theory. *ACM Trans. Inter. Tech.*, 5(4):660–704, 2005.
28. K. Opocenska and M. Kopecky. Incox – a Language for XML Integrity Constraints Description. In *DATESO'08*, pages 1–12. CEUR-WS.org, 2008.
29. Y. Papakonstantinou and V. Vianu. DTD Inference for Views of XML Data. In *PODS'00: Proc. of the 19th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems*, pages 35–46, New York, NY, USA, 2000. ACM.
30. K. E. Shafer. Creating DTDs via the GB-Engine and Fred. In *SGML'95: Conf. Proc.*, page 399. Graphic Communications Association, 1995.
31. J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and

Opportunities. In *VLDB'99: Proc. of the 25th Int. Conf. on Very Large Data Bases*, pages 302–314, San Francisco, CA, USA, 1999. Morgan Kaufmann.
32. H. Shiu, J. Fong, and R. P. Biuk-Aghai. Reverse Engineering XML Documents Into DTD Graph With SAX. *WSEAS Transactions on Computers*, 5(6):1236–1241, 2006.
33. H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. *XML Schema Part 1: Structures (Second Edition)*. W3C, 2004. `http://www.w3.org/TR/xmlschema-1/`.
34. O. Vosta, I. Mlynkova, and J. Pokorny. Even an Ant Can Create an XSD. In *DASFAA'08: Proc. of the 13th Int. Conf. on Database Systems for Advance Applications*, LNCS, pages 35–50. Springer, 2008.
35. R. K. Wong and J. Sankey. *On Structural Inference for XML Data*. Technical Report UNSW-CSE-TR-0313, School of Computer Science, The University of New South Wales, 2003.