Schema Extraction from XML Data: A Grammatical Inference Approach

Boris Chidlovskii
Xerox Research Centre Europe, Grenoble Laboratory
6, Chemin de Maupertuis, F–38240 Meylan, France
childovskii@xrce.xerox.com

Abstract. New XML schema languages have been recently proposed to replace Document Type Definitions (DTDs) as schema mechanism for XML data. These languages consistently combine grammar-based constructions with constraint- and pattern-based ones and have a better expressive power than DTDs. As schema remain optional for XML data, we address the problem of schema extraction from XML data. We model the XML schema as extended context-free grammars and propose the schema extraction algorithm that is based on methods of grammatical inference. The extraction algorithm copes also with the schema determinism requirement imposed by XML DTDs and XML Schema languages. We report the tests result of schema extraction on a collection of real XML documents.

1 Introduction

The Extensible Markup Language (XML) has emerged as a new standard for exchange and manipulation of structured documents. The XML format is self-describing; XML documents comprise hierarchically nested collection of elements, where each element can be either atomic or complex, i.e, a sequence of nested sub-elements. The internal structure of XML documents is described by Document Type Definitions (DTDs) which are the de facto standard XML scheme language [28]. DTDs provide basic capabilities for defining the element contents in the form of regular expressions. Validation of an XML document is reached by matching of element content strings against element content models in a DTD. The knowledge of XML schema is difficult to underestimate; it plays a principal role in the XML data management, including the query formulation, query planning and storage [10, 27].

Initially designed for document management applications, DTD capabilities appear to be limited, in both syntax and expressive power, for wider application domains. As result, a number of new XML schema languages, including DSD, Schematron and XML Schema, have been recently proposed by the XML community [8, 16, 20]. Beyond using XML syntax, these languages extend the DTD model with novel important features, such as simple and complex types, rich datatype sets, occurrence constraints and inheritance. Features of the new schema proposals have been compared and classified in [18]. All languages initially follow a certain conceptual approach, such as grammar-based one in XML Schema or pattern-based in Schematron. However, most of them recognize the importance of integrating different constructions and often propose a multi-conceptual set of features. As an example, XML Schema language extends the grammatical constructions with a large set of constraints and patterns [5, 8, 14].

Problem statement. Despite the importance of schema information, schema definitions are not obligatory in XML documents. This raises the *problem of extracting the schematic information from XML documents*. The extraction often requires certain generalization of input data; in the ideal case, the extracted schema should, one one side, *tightly represent* the data, and be *concise and compact*, on the other side. As the two requirements essentially contradict each other, finding an optimal tradeoff is a difficult and challenging task.

Among existing conceptual approaches to the XML schema, the grammar-based one seems to be more promising for the schema extraction, as it can benefit from a multitude of methods developed by the grammatical inference community [4, 30]. In our approach, we do follow the grammar-based schema extraction and we will align our algorithm with the XML Schema language.

Automatic schema extraction can be highly helpful in a number of situations. First, it can be used for real world *XML data with complex structure*; for such data, DTD design is a difficult and error-prone work that often results in a large number of badly designed DTDs [24]. Second, the automatic

schema extraction can be beneficial for in mediator and information integration systems that cope with *heterogeneous collections* of XML documents and may have a need in a common schema for all documents. Finally, the automatic schema extraction may assist the designer in the schema definition. It can consist in analysis of available (possibly partial) XML data and refining, that is, constraining or loosing, the existing schema patterns and finding new ones.

1.1 Our contribution

Schema formalism. We adopt the XML schema formalism based on *extended context-free grammars* (ECFG). ECFGs have been already used as XML schema mechanism in [21, 29]. We extend this model by allowing *range regular expressions* in nonterminal productions; such regular expressions combine grammatical forms and constraints for nonterminals and element groups. We establish the one-to-one correspondence between components of ECFGs and constructions of XML Schema language.

Schema inference problem. For the proposed schema formalism, we address the problem of schema extraction from XML data. The ECFG-based schema model makes the extraction problem more complex than in the DTD extraction case; the former is reduced to the inference of context-free grammars, while the latter is equivalent to the inference of a (finite) set of regular grammars.

We identify three important components of the schema extraction problem, namely, (1) *induction the context-tree grammars* from XML documents represented as structured examples, (2) *generalization of content strings* into regular expressions, and (3) *constraining datatypes* for simple XML elements. The second problem is the same as with the DTD extraction, but the first and third ones are relevant to the powerful schema mechanisms offered by novel XML schema languages.

For the first problem, we adopt and extend the method of CFG inference from structural examples [26]. For the third problem (datatypes constraining), we develop an algorithm based on the subsumption relationship among elementary datatypes in XML Schema language [5]. Finally, for the second problem (content generalization) we propose a solution alternative to those used for DTD extraction [7, 12]; it copes with the determinism requirement and the occurrence constraints in XML Schema language.

Determinism. Both XML DTDs and XML Schema require an easy validation of XML data against corresponding schema [14, 28]. Such an ease is defined as "determinism" and closely corresponds to the notion of unambiguity in the formal language theory [15]. Determinism can essentially constrain the power of ECFG model, as a large part of grammars does not provide the feature. We study the impact of the determinism requirement on the ECFG model and the schema extraction problem. We distinguish between *horizontal and vertical determinism* that address the ease of vertical and horizontal navigation in an XML document tree. We study both types of the determinism and develop rules for inferring the deterministic ECFG schema from XML data.

1.2 State of art

Semistructured data and schema. XML data model is a modification of the semi-structured data model [22]. For semi-structured data, schema may have the form of Data Guides [13] that represent a condensed graph-like representation of semi-structured data. A Data Guide can be obtained from the semi-structured data by removing element duplications and repeating paths. An alternative approach consists in finding the most frequent structural patterns in a semi-structured database. The structural patterns in so-called "road maps" approach [17] can be both cyclic and acyclic, and they can overlap. The method proposed in [17] discovers the set of *maximally frequent* patterns, that is, each pattern in the set occurs more than a given minimal number m and is not contained that any other frequent pattern.

DTD limitations and CFGs. The XML data model extends the semi-structured data model by imposing the order on the appearance of elements and their sub-elements. Inherited from SGML, XML DTDs have been designed for document management applications. Most straightforward limitations

of DTDs are a non-XML syntax, a limited set of datatypes and loose structured constraints [19, 21]. Moreover, DTDs are limited from the point of view of XML querying, as no *tight* DTDs can be induced even for view queries [21]. The possible solution to the tightness problem requires the extension of DTDs with the sub-typing and specialization, it makes the resulting schema model close in expressive power to the XML Schema language.
DTDs and five novel XML schema languages [8, 16, 20] have been recently compared in [18]. It classifies all important features for defining XML schema, such as simple and complex types, elementary datatypes, constraints for elements/attribute values and appearance, inheritance, and compares how all these languages support them. While DTDs appear to have the weakest expressive power, the most powerful languages are XML Schema [8], Schematron [16] and DSD languages [20].

DTDs extraction from SGML/XML. Initially, systems for extracting DTDs have been designed for SGML [1, 3, 11], they propose a quite simple and straightforward extraction methods. Later, some of these systems [11] have been extended to XML [7].
The most advanced algorithm for extracting DTDs from XML documents is proposed in the XTRACT system [12]. The extraction algorithm represents a sequence of sophisticated induction steps in order to induce concise and intuitive DTDs. It finds sequential and choice patterns in the input sequences, in order to generalize and factor them, then it applies the Minimal Description Length principle to find the best DTD among the candidates.

Inference of regular and context-free grammars. Grammatical inference is a well-established domain in machine learning and artificial intelligence. Though the inference problem is found undecidable in the general case for both regular and context-free grammars, the grammatical community has developed a number of sophisticated inference methods and identified a number of language subclasses for which the inference is decidable [4, 30].
For the schema extraction algorithm, we re-use induction methods developed for context-free and regular grammars. First, an XML data is equivalent to a structured example in the grammatical inference theory, (structured example is a derivation tree of a CFG with removed nonterminal labels). It makes possible to reuse the CFG inference from structured examples studied in [25, 26]. Second, we use regular grammar inference methods [4] in order to generalize content strings into regular expressions.
In the following, we assume familiarity with basic notions of the language theory, including regular and context-free grammars, finite-state automata and parsing ambiguity [15].
The remainder of the paper is organized as follows. Section 2 introduces the ECFG schema formalism and the schema induction algorithm. Section 3 analyzes and develops methods for the nonterminal merge during an ECFG inference. Determining datatypes for simple elements is discussed in Section 4. In Section 5, we show how to generalize content strings into range regular expressions. Section 6 reports the inference experiments on real XML data and Section 7 concludes the paper.

2 Extended context-free grammars as XML schema

2.1 XML example

As example, we consider an XML data about European soccer clubs.[1] It contains information about each club, including the name and successes in two European tournaments, Champion Leagues and UEFA.

```
<teams>
   <team><name>Juventus</name>
      <ChLeague>
         <year>1999</year><result>semi-final</result>
         <year>1997</year><result>final</result>...// 5 successes in total
      </ChLeague>
```

[1] Data available at `http://www.chez.com/eurofoot/`

```
      <UEFA>
         <year>1995</year><result>final</result>...// 7 successes in total
      </UEFA>
   </team>
   <team><name>Manchester United</name>
      <ChLeague>
         <year>1999</year><result>winner</result>...// 4 successes in total
      </ChLeague>
   </team>
   <team><name>Hertha Berlin</name></team>
... // 46 teams in total
<teams>
```

XML Schema language. XML Schema language [5, 8, 14] offers a powerful feature set for defining the internal structure of XML documents. For the goal of schema extraction, we consider an important subset of the language features willing to associate them to components of extended context-free grammars. Here the features we address:

- *elementary datatypes* for simple elements (with no sub-elements),
- *complex types* for complex elements (with sub-elements),
- *sequence* and *choice* groups of elements,
- *occurrence constraints* for elements and groups (`MaxOccurs`, `MinOccurs`.).

Below is the XML Schema definition for the soccer data; it includes the initial element `teams`, the complex type `TeamType` for `team` elements, and the complex type `ListType` for `ChLeague` and `UEFA` elements; `MinOccurs` and `MaxOccurs` facets constrain the occurrence ranges.

```
<element name='teams'>
     <complexType>
          <element name='team' type='TeamType' maxOccurs='500'/>
     </complexType>
     <complexType name='TeamType'>
          <element name='name' type='String'/>
          <element name='ChLeague' type='ListType' minOccurs='0' maxOccurs='1'/>
          <element name='UEFA' type='ListType' minOccurs='0' maxOccurs='1'/>
     </complexType>
     <complexType name='ListType'>
          <group minOccurs='1' maxOccurs='100'>
          <element name='year' type='PositiveInteger'/>
          <element name='result' type='String'/></group>
     </complexType>
</element>
```

In the following sections, we consider the equivalent ECFG and introduce the schema extraction algorithm.

2.2 Extended context-free grammars

We model XML schema as range extended context-free grammars. ECFGs have been already used as XML schema model in [21, 29]. We adopt the model and extend it with range constraints for both elements and groups in a schema definition.

Range regular expression is a regular expression over an alphabet Σ, where each term is defined with the range $[l : r]$ of possible occurrences, where l and r are nonnegative integers, $0 \leq l \leq r$, r can be ∞. Using the range notation, the Kleen closure $a*$ is equivalent to $a[0 : \infty]$, and $a+$ and $a?$ are equivalent to $a[1 : \infty]$ and $a[0 : 1]$. For simplicity, we abbreviate the range $[l : l]$ as $[l]$ and omit the range $[1]$. As an example, regular expression $a(bc+)*$ will be written as $a(bc[1 : \infty])[0 : \infty]$ in this notation.

Range regular expressions have the same expressive power as regular expressions [29], however there is a clear benefit of using them in the schema formalism. On one side, range regular expressions extend the schema model with occurrence constraints in the same way as most XML schema languages do. On the other side, it prompts the definition of *limited element occurrences* instead of unlimited ones in DTDs ($a*$ and $a+$), which is highly valuable for query formulation and optimization of XML data storage [10, 27].

A *(range) extended context free grammar* (ECFG) is defined by 5-tuple $G = (T, N, D, \delta, Start)$, where T, N and D are disjoint sets of terminals, nonterminals and datatypes; $Start$ is an initial nonterminal and δ is a finite set of production rules of the form $A \rightarrow \alpha$ for $A \in N$, where α is a range regular expression over *terms*, where one term is a terminal-nonterminal-terminal sequence like $t\ B\ t'$, where $t, t' \in T$ and $B \in N \cup D$. The notion of term is introduced here to capture the well-formedness of XML [28]. In the following, we will abbreviate a term $t\ B\ t'$ as $t : B$.

XML Schema and ECFGs The ECFG-based schema model addresses the chosen subset of features in the XML Schema language in such a way that any XML Schema definition corresponds to some extended context free grammar and vice versa. The correspondence between an ECFG G and XML Schema definition S is the following. The terminal and datatype sets T and D in G correspond to the sets of element names and elementary datatypes in S, respectively. The nonterminal set N in G corresponds to the set of complex types in S, both named and abstract. Finally, productions in G corresponds to definitions of the complex types. One production is a *sequence* or *choice* group of typed elements or other (nested) groups. The role of *Start* nonterminal in ECFG can play either element of named complex type definition. Table 1 summarizes the correspondence between components of XML Schema definitions and extended CFGs.

XML Schema language	ECFG
Element name (tag)	terminal
Element definition	production
Elementary datatype	datatype
Named complex type	non-terminal
Abstract complex type	non-terminal
Complex type definition	one or more productions
Sequence element group	sequential pattern in a production
Choice element group	disjunction pattern in a production

Table 1. Correspondence between XML Schema features and ECFG components.

Example 1. Consider the XML Schema definition for the soccer XML data in Section 2.1. The corresponding ECFG G is given by $G = (T, N, D, \delta, Start)$, where T={`teams, team, name, ...`}, N={*Start, TeamType, ListType*}, D={**String, PositiveInteger**}, and δ contains the following production rules:

Start → `teams:`*TeamsType*
TeamsType → `(team:`*TeamType*`)[0:500]`
TeamType → `name:String (ChLeague:`*ListType*`)[0:1] (UEFA:`*ListType*`)[0:1]`
ListType → `(year:`**PositiveInteger** `result:`**String**`)[1:100]`

DTDs as ECFGs. DTDs is a particular case of ECFGs where one nonterminal is assigned to one terminal (given by the corresponding `<ELEMENT...>` definition), and there exists one-to-one correspondence between sets N and T. Therefore, when extracting DTDs from XML data, the nonterminal set N is directly reconstructed from the terminal set T, and the inference problem is reduced to the generalization of content strings for each nonterminal into a regular expression [12].

2.3 Determinism and extended context free grammars

By both XML DTDs and XML Schema requirements [14, 28], parsing and validating an XML document with a schema should be deterministic, that is, validating each item in the document "can be uniquely determined without examining the content or attributes of that item, and without any information about the items in the remainder of the sequence"[14].

It turns out that the determinism can essentially constrain the power of ECFG model, as far not every ECFG satisfies the requirement. As consequence, any schema extraction algorithm that does not consider such important constraint, would provide rather a partial solution.

Assume a parser validates an XML document and observes an opening tag $< b >$ contained in the element a. By the requirement, two things should be determined with one-token lookahead. The first one is the rule for processing the content of element b and the second one is the valid particle for b in the content model/regular expression of a. Thus, we distinguish between *vertical* and *horizontal* determinism, as they address the vertical and horizontal navigation through an XML data tree [19]. Here we consider the vertical determinism, and the horizontal determinism will be discussed in Section 5.

The *vertical determinism* requires that at any complex element, the rule for any sub-element should be determined with one-token lookahead. Formally, we call two terms $t : A$ and $t' : A'$ *ambiguous* if $t = t'$ but $A \neq A'$. Ambiguous terms in productions make difficult parsing and validating XML documents [2, 29]. For example, in the production $A = t : A' \mid t : A''$, the correct choice between the first and second terms will require lookahead at least 2 and further analysis of productions for A' and A''. On the other hand, the absence of ambiguous terms in productions guarantees the vertical determinism of an ECFG and we will be using this as the *sufficient* condition for inferring deterministic ECFGs.

Preposition 1 *An ECFG G guarantees the vertical determinism if no production in G contains ambiguous terms.*

2.4 XML as structured example

For the need of schema inference, we represent XML documents as a *structured example* of an (unknown) ECFG. Here, structured example is a derivation tree of a CFG with all nonterminal labels removed [26]. Due to the XML well-formedness, it is straightforward to represent XML documents as structured examples; in any such tree, each pair of opening and closing element tags indicates the begin and end of the element content subtree.

Structured example for the soccer data is given in Figure 1. Internal nodes (empty circles) correspond to the complex elements, while leaves can be either tags (they drive the tree construction) or simple elements (filled circles).

Presented in this way, we reduce the schema extraction from XML documents to the ECFG inference from structured samples, which in turn is split into two parts, the inference of complex types for complex elements and datatypes for simple elements.

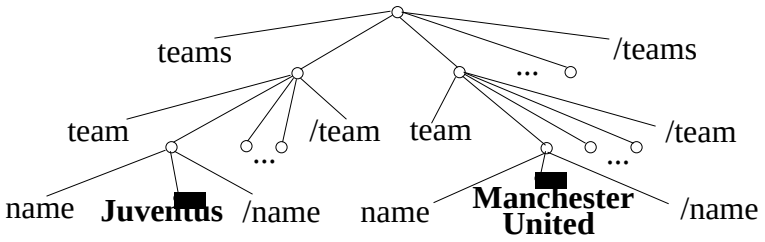

Fig. 1. Example XML data as structural example.

2.5 Schema induction algorithm overview

The extraction algorithm comprises several important steps:

Algorithm 1. Schema extraction from XML data.
0. Represent XML documents as set I of structured examples.
1. Induce an extended context-free grammar G from I:
   1.1. Create the initial set of nonterminals N;
   1.2. Merge nonterminals in N with the similar content and context;
   1.3. Determine tight datatypes for terminals in G;
   1.4. Generalize content sets in nonterminal productions in range regular expressions.
2. Transform the result ECFG G into an XML Schema definition S.

Below, we explain the work of Algorithm 1 on the soccer example data. We start by generating the initial set of nonterminals. We assign the generic datatype **Any** [2] to all simple elements and label each complex element in the XML tree with a unique nonterminal. We generate productions for nonterminals as contents of corresponding nodes. For the soccer data, the initial nonterminal set N has the following productions:

Step 1.1.
Start $\rightarrow$ `team:`A_1 `team:`A_2 `team:`A_3 `...(`*46 teams in total*`)`
A_1 $\rightarrow$ `name:`**Any** `ChLeague:`A_{47} `UEFA:`A_{48}
A_2 $\rightarrow$ `name:`**Any** `ChLeague:`A_{49}
A_3 $\rightarrow$ `name:`**Any**
`...`
A_{47} $\rightarrow$ `year:`**Any** `result:`**Any** `... // (`*7* `year-result` *pairs in total*`)`
A_{48} $\rightarrow$ `year:`**Any** `result:`**Any** `... // (`*5 pairs*`)`
A_{49} $\rightarrow$ `year:`**Any** `result:`**Any** `... // (`*4 pairs*`)`
`...`

The second step is the merge of nonterminals. As the production for $Start$ contains ambiguous terms with terminal `team`, we merge nonterminals A_1, A_2, A_3 as ones with *similar context*. The result of merge contains ambiguous terms with terminals `ChLeague`, thus we merge nonterminals A_{47} and A_{49}. It gives the following set of productions:

Step 1.2.a.
Start $\rightarrow$ `team:`A_1 `team:`A_1 `team:`A_1 `...(`*46 teams in total*`)`
A_1 $\rightarrow$ `name:`**Any** `ChLeague:`A_{47} `UEFA:`A_{48}
A_1 $\rightarrow$ `name:`**Any** `ChLeague:`A_{47}
A_1 $\rightarrow$ `name:`**Any**
A_{47} $\rightarrow$ `year:`**Any** `result:`**Any** `... // (`*7 pairs*`)`
A_{48} $\rightarrow$ `year:`**Any** `result:`**Any** `... // (`*5 pairs*`)`
A_{47} $\rightarrow$ `year:`**Any** `result:`**Any** `... // (`*4 pairs*`)`
`...`

Next, we observe that nonterminals A_{47} and A_{48} have repeating pairs of elements `year` and `result`, and we merge them as ones with *similar content*. As result, we obtain the ECFG where productions have right parts in the form of disjunctions of different contents.

[2] Actually, **Any** is equivalent to the primitive datatype `String`.

Step 1.2.b.

Start $\rightarrow$ `(team:`A_1`)[46]`

A_1 $\rightarrow$ `name:`**Any** `ChLeague:`A_{47} `UEFA:`A_{47}`| name:`**Any** `ChLeague:`A_{47}`|...`

A_{47} $\rightarrow$ `(year:`**Any** `result:`**Any**`)[7] | (year:`**Any** `result:`**Any**`)[5] |...`

We generalize the production for A_1 by factoring the term `name`:**Any** and detecting that `ChLeague` and `UEFA` elements can occurs 0 or 1 times. Then, rewriting the production for A_{47} requires the generalization of all contents in one range regular expression. Assume that we collect trophy contents for all 46 clubs (63 items in total) and the longest content reports 8 successes in a competition. The *tight generalization* for A_4 is therefore (`year`:**Any** `result`:**Any**)[1:8].
Finally, we analyze the values of simple elements `name`, `year` and `result` and induce that `year` element has all integer values and it is valuable to constrain its type to **unsignedShort**. For elements `name` and `result`, **Any** is replaced with the datatype **String**. The final ECFG for soccer XML data is the following:

Step 1.4.

Start $\rightarrow$ `(team:`A_1`)[46]`

A_1 $\rightarrow$ `name:`**String** `(ChLeague:`A_4`)[0:1] (UEFA:`A_4`)[0:1]`

A_4 $\rightarrow$ `(year:`**UnsignedShort** `result:`**String**`)[0:8]`

Comparing the resulting ECFG to the initial one in Section 2.1, one can observe the minimal difference, it concerns mainly the occurence ranges.
In Algorithm 1, Step 1.1 (initial nonterminal set) is straightforward and Step 1.4 (content generalization) is shared with the DTD extraction. Two other important steps, the merge of nonterminals and determining datatypes for simple elements are specific to the ECFG schema model only. In the following sections we describe and study the two steps in detail.

3 Nonterminal merge

We assume the initial set N of nonterminals is built from a set I of structured examples. When a context-free grammar is infered from structural examples [26], the induction is conducted by merging nonterminals with *equivalent content* and *equivalent context* as follows:
1. Content equivalence: $A \rightarrow \alpha$ and $B \rightarrow \alpha$ in δ implies that $A = B$,
2. Context equivalence: $A \rightarrow \alpha B \beta$ and $A \rightarrow \alpha C \beta$ in δ implies that $B = C$, $\alpha, \beta \in (N \cup D)^*$.
In the case of ECFGs, these two rules should be properly extended. First, they should cope with the term-based structure of productions and the determinism requirement. Second, the content equivalence condition appears to be too strong as it fails to identify and merge nonterminals whose right parts are two different instances of one regular expression (like nonterminals A_{47} and A_{48} at Step 1.2.a). Therefore, we propose to replace it with a weeker condition of *silimarity*.
For the determinism requirement, we implement the condition established in Section 2.3 for the vertical determinism, it disallows ambiguous terms in any grammar production. This gives us the following generalization of the equivalent context rule (2):
3. Vertical determinism (or context similarity): $A \rightarrow \alpha$ *t:B* β *t:C* γ implies that $B = C$.

Merge by similar content. The merge of productions with ambiguous terms described above is obligatory and imposed by the determinism requirement. Instead, the merge of nonterminals with similar content may be optional and requires the introduction of a metric that can control the merge accordingly to the user's perception.
Consider again the soccer example and denote elements `year`, `result` and datatype `string` as y, r and s, respectively. Then non-terminals $A_{47} \rightarrow$ (*y:s r:s*)[7] and $A_{48} \rightarrow$ (*y:s r:s*)[5], are likely that their right parts are instances of the range regular expression (*y:s r:s*)[*l:r*] and therefore the user may want to merge A_{47} and A_{48}. Below we propose a method for merging nonterminals with different but similar contents.

We consider two contents as strings over terminal alphabeth T and thus we can use the vector space model (VSM) to measure the string similarity [9]. We generalize the VSM model by considering n-grams in content strings, where n-gram is a sequence of n consequent elements in the content, $n = 1, 2, 3, ...$ We denote the set of n-grams in a content string s as $P_n(s)$. If the terminal set T has k elements, then there may be up to $O(k^n)$ n-grams. We count n-grams for the content string c and represent c as a normalized vector $c = \{c^i\}$, $i = 1, 2, \ldots$ of n-gram frequencies, $\sum c^i = 1$.
The *similarity measure* between two content strings c_1 and c_2 is given by $\mathcal{M}(c_1, c_2) = \sum c_1^i \cdot c_2^i$ which is interpreted as the cosine of the angle between c_1 and c_2 in the multi-dimensional space. Since each content vector is normalized, then $\mathcal{M}(c_1, c_2)$ is normalized too. Two contents c_1 and c_2 are considered *similar* if $\mathcal{M}(c_1, c_2) > th$, where $0 \leq th \leq 1$ is a threshold value set by user. The choice of th is important; values close to 0 may result in merging most of nonterminals, while values close to 1 make difficult the merge of even visibly similar contents. As experiments show, the optimal tradeoff is reached when th values are $[0.3, 0.7]$ range.
The similarity between two content strings can be generalized to the similarity between nonterminals whose right parts are disjunctions of contents. If nonterminals are given by two disjunction of contents, then their similarity is given by the maximal similarity over pairs of contents from corresponding conjunctions. Formally, if two nonterminals $A \rightarrow \alpha$ and $B \rightarrow \beta$ are such that $\alpha = c_1|c_2|\ldots$ and $\beta = c'_1|c'_2|\ldots$, then

$$\mathcal{M}(\alpha, \beta) = max_{c_i \in \alpha, c'_j \in \beta} \mathcal{M}(c_i, c'_j).$$

This allows us to rewrite the strong content equivalence condition (1) with a weaker similarity one:
4. Content similarity: For $A \rightarrow \alpha$ and $B \rightarrow \beta$ in Σ, such that M$(\alpha, \beta) > th$, implies that $A = B$.

4 Datatype inference

All simple elements in structured examples of I are initially labeled with generic datatype **Any** in order to simplify the ECFG inference. Once the nonterminal merge is completed, we return to simple elements in the induced ECFG and constrain properly their datatypes. For each simple element e in grammar productions, we identify all values of e in the collection I and analyze this value set V against the datatype set D in order to induce the *minimal required* datatype which contains all values of V.
Let d and d' be two different elementary datatypes in D. We say that datatype d *subsumes* d', denoted $d \supset d'$, if any value of d' (actually, the string representing the value) is also a value of datatype d. Then, datatype d *tightly subsumes* d', denoted $d \succ d'$, if d subsumes d' and there is no other datatype d'' such that $d \supset d'' \supset d'$.
Datatypes in set D form a directed acyclic graph G_D called *datatype subsumption graph*. Each datatype $d \in D$ has a corresponding node in G_D, and a directed arc from d to d' in G_D indicates that $d \succ d'$.
XML Schema language supports a rich set of datatypes, including many numeric datatypes [5]. We analyze the datatype definitions in XML Schema language and build the corresponding datatype coverage graph (see Figure 2). In the graph, only datatype **String** has no incoming arcs, as there is no datatype $d \in D$ such that $d \succ \mathbf{String}$.
Consider a simple element e in an inferred ECFG schema and its value set V. The *minimally required* datatype for e is the smallest datatype in graph G_D which accommodates *all* values of V. Consider the element **year** in our soccer example. It has integer values in the range 1960-1999 and any numeric datatype of XML Schema language can accommodate these values. The smallest among these datatypes is **unsignedShort** and we choose it as the minimally required datatype for **year** element.
We have implemented the datatype inference by using descriptions for all numeric datatypes in XML Schema language [5]. Description for a datatype d is given by a pair (*lexical rule, range*), where *lexical rule* is a lex-like regular expression and *range* is a numeric range constraint. String s is a valid value of d if it matches the lexical rule of d and satisfies its range constraint. The table below reports descriptions for some of implemented numeric datatypes.

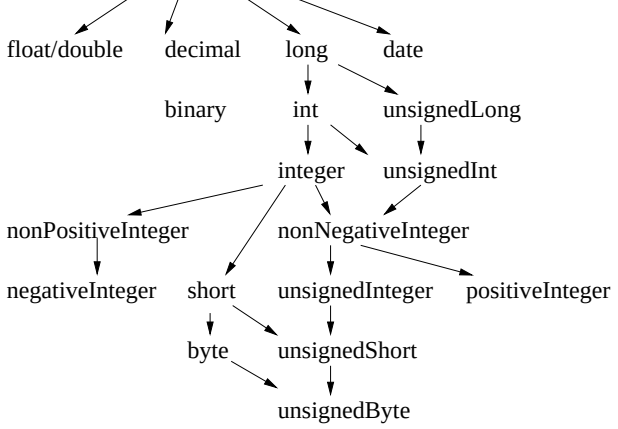

Fig. 2. Datatype subsumption graph for XML Schema language.

Datatype	Lexical rule	Range
decimal	[-]?[1-9][0-9]*(.[0-9]*[1-9])?	[-∞, ∞]
integer	[-]?[1-9][0-9]*	[-∞, ∞]
NegativeInteger	[-]?[1-9][0-9]*	[-∞, 1]
short	[-]?[1-9][0-9]*	[-32768,32768]
ByteUnsigned	[1-9][0-9]*	[0,126]

5 Content generalization

The content generalization step in Algorithm 1 addresses the same goal as the DTD extraction systems, and any of existing methods [3, 7, 12] could be adopted here. However, two important differences lead us to proposing an alternative solution. First, unlike the DTD extraction, all generalizations for finite contents are controlled by finite occurrence ranges for elements and groups, therefore, any tight generalization should be a *finite regular language*.

Second, we take into consideration the *horizontal determinism* requirement imposed by both DTDs and XML Schema languages [14, 28]. While vertical determinism requires one-token lookahead identification of rules for sub-elements, the horizontal determinism refers to the one-token lookahead determination of rules for elements in a content model. For example, two regular expressions $ab|ac$ and $(ac)*a$ are both ambiguous as for any valid string $s = as'$, disambiguation of the starting element a (choice in the first example and the sequence in the second one) would require looking at the second element in the string.

Deterministic (or one-unambiguous) content models have been studied in [6] where a formal procedure for the verification of regular expression determinism has been proposed. The verification consists in building corresponding finite-state automaton and testing so-called *orbit* property. If the automaton satisfies the orbit property, the initial regular expression is deterministic. We re-use the results from [6] to the case of inferring deterministic regular expressions from string sets.

The input to the generalization step is a given nonterminal production, which is usually a disjunction of contents. As these contents do not have conflicting terms, we omit nonterminals from contents and consider the content disjunction as a set S of terminal strings, $S = \{s|s \in T^*\}$. Our algorithm of content generalization contains four steps:

Algorithm 2. Content generalization.
1. Generalize the set S in the form of finite-state automaton;
2. Convert if necessary the automaton into deterministic one;
3. Convert the result automaton into corresponding regular expression;
4. Instantiate the result regular expression by tightening the ranges for all Kleen stars.

Step 1. Automata generalization. First, we generalize the sample set S in the form of finite-state automaton. We build a finite-state automaton $FA(S)$, called *follow automaton*, from the set $P_2(S)$ of all 2-grams over the strings from set S (see Section 3). Such automaton has $l + 1$ nodes (initial *Start* state and one state for each symbol in Σ); $FA(S)$ has a transition from state a to state b if and only if $ab \in P_2(S)$. If symbol a is the first symbol in string $s \in S$, then $FA(S)$ has a transition from the *Start* state to state a. If symbol a is the last symbol in strings $s \in S$ the corresponding state is marked as final state.
It is important to note that $FA(S)$ provides the tight and compact generalization for the content model of S. However, it may not satisfy the orbit property and therefore no deterministic regular expression can be found for it.

Example 2. Consider the following example $S = \{acbb, acbcbb, cbbb, cbcbb\}$. We have $P_2(S) = \{ab, ac, bb, bc, cb\}$ and the corresponding $FA(S)$ automaton is given in Figure 5.a).

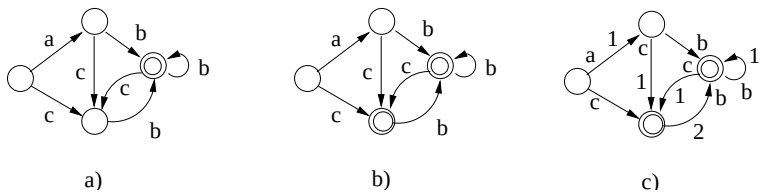

Fig. 3. Automata generalization for the sample set $S = \{acbb, abbcbb, cbbb, cbcbb\}$. a) $FA(S)$, b) $dFA(S)$, c) Instantiation $dFA(S)$ with $acbcbb$.

Step 2. Generalizing $FA(S)$ into deterministic one. If the follow automaton $FA(S)$ does not satisfy the orbit property, we further generalize it, by adding additional transitions or labeling some states as final ones in the way that the result automaton, denoted $dFA(S)$, does satisfy the orbit property. Essentially, the orbit property requires that for any cycle in the automaton, all states in the cycle are *gates*, that is, they all are either final or not final, and they all have the same preceding and following states [6].
Thus we detect the minimal set of additional transitions and state changes in $FA(S)$ that generalize it into deterministic automaton. We omit details of the algorithm here. Automaton $FA(S)$ in Figure 5.a is generalized into $dFA(S)$ by marking the state for c as final, see Figure 5.b. Automaton $dFA(S)$ satisfies the orbit property and corresponds to the deterministic regular expression $a?(b|c)*$.
Step 3. Conversion a finite-state automata into regular expression. We use the Grail software [23] which contains the `fmtore` routine for converting automaton $dFA(S)$ into deterministic regular expressions $dRE(S)$.
Step 4. Instantiation of unambiguous regular expression. We finish the content generalization procedure by initiating the (infinite) regular expression $dRE(S)$, that is, by replacing all its *- and +-ranges with tight (finite) occurrence ranges. To instantiate a deterministic regular expression with a set S of input strings, we reuse the deterministic automaton $dFA(S)$, obtained at Step 2. For a given string s from$S$, the instantiation consists in (1) traversing the automaton with s, 2) marking all occurrences of traversing any edge in the automaton and 3) detecting the minimal and maximal occurrences for the corresponding range.
We associate each transition in $dFA(S)$ with corresponding particle in the $dRE(S)$. As an example, consider the instantiation of deterministic regular expression $dRE(S) = a?(b|c)*$ with string $s = acbcbb$. We traverse $dRE(S)$ with s and count how much each transition is traversed, the instantiation of $dRE(S)$ with $s = acbcbb$ is given in Figure 5.c. It corresponds to the range regular expression $a(c|b)[5]$. Having collected the instantiations for all strings in $S = \{acbb, acbcbb, cbbb, cbcbb\}$, we obtain the result tight range regular expression $a[0:1](c|b)[2:5]$.

6 Experiments

We have conducted a series of experiments to test the schema extraction algorithm proposed in Sections 3-5. For tests, we have used four collections of real XML documents (with their DTDs), namely, collections of Shakespeare plays, religious works and astronomy files available at `http://www.goxml.org/` and the Sigmod Record archive (available at `http://www.dia.uniroma3.it/Araneus/Sigmod/`).
We apply the schema extraction algorithm to all documents in collections. Our first goal is to test the nonterminal merge methods (see Section 3) with the threshold value th for content similarity varying in the range [0,1] and compare the result schema to corresponding DTDs. The value of th value controls the generalization of input data; th=1 disallows merges by the content similarity, while th=0 would merge nonterminals having at least one common sub-element (or n-gram in the general case).
Figure 6 shows the performance of the nonterminal merge for sample documents: `1021A.xml` from the astronomy collection and `sigmod.xml` in Figure 6.a, `hamlet.xml` from Shakespeare collection and `bom.xml` from the religion collection in in Figure 6.b. The figures plot the number of nonterminals in result ECFGs for th values in [0,1] range as well as the number of `ELEMENT` definitions in the corresponding DTDs.
The right extreme of plots (th = 1.0) shows the pure performance of the context similarity merge, imposed by the vertical determinism requirement. As the plotting shows, the vertical determinism provides a very satisfactory reduction of the nonterminal number, comparable to the size of corresponding DTDs. In the case of `bom.xml` file, the result size (22 nonterminals) is even lower, for the file contains only 20 from 28 elements defined in the DTD.

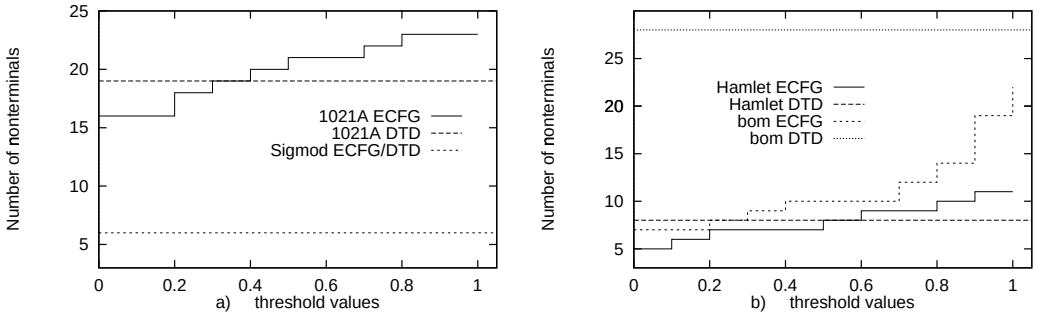

Fig. 4. Nonterminal merge for sample documents; a) `1021A.xml` and `sigmod.xml`, b) `hamlet.xml` and `bom.xml`.

Decreasing threshold values toward 0 makes easier the merge of remaining nonterminals and can further reduce their number; the reduction is particularly essential for `bom.xml` file where 8 elements appear to have the equivalent definition `(PCDATA|<i>)*`. An interesting exception is represented by `sigmod.xml` file, where the vertical determinism has reached the maximal reduction and no additional merge is possible with the content similarity, whatever the threshold value is.
Table 2 summarizes the results of schema extraction for all tested XML documents, it equally reports the datatype constraining for simple elements. The table confirms a good performance of the context similarity merge for documents which vary in both size and the initial number of nonterminals. Abbreviations used in the table are the following:

- SE - number of simple elements in a document,
- SE_c - number of simple elements with constrained datatypes, $SE_c \leq SE$,
- CE - number of complex elements in a document,
- N_i - initial number of nonterminals,
- N_f - final number of nonterminals.

To check the scalability of the schema extraction algorithm, we test it, in addition to separate documents, on entire Shakespeare and religious collections (8 and 4 files respectively), and compare

SigmodRecord	sigmod.xml	7	4	6	3143	6	6	6
Shakespeare	caesar.xml	10	0	8	836	5	9	12
	hamlet.xml	10	0	8	1204	6	8	11
	lear.xml	10	0	8	1125	5	9	11
	macbeth.xml	10	0	8	701	5	9	12
	merchant.xml	10	0	8	675	6	9	13
	othello.xml	9	0	7	1227	5	9	12
	r_and_j.xml	10	0	8	891	5	9	13
	titus.xml	10	0	8	806	5	10	12
	all 8 files	10	0	8	7465	6	13	14
Religion	bom.xml	1	0	21	504	7	10	22
	nt.xml	1	0	14	293	7	9	18
	ot.xml	1	0	20	998	8	11	25
	quran.xml	1	0	15	132	6	9	21
	all 4 files	1	0	28	1927	8	14	31
Astronomy	1005.xml	17	2	19	38	16	19	21
	1021A.xml	17	0	19	388	16	21	23
	1061B.xml	18	1	25	71	17	24	26
	1062C.xml	18	2	21	99	16	22	22
	1099.xml	18	0	19	87	16	20	23
	1008.xml	17	1	21	99	15	21	22
	1110.xml	18	2	19	67	16	19	20
	1112.xml	17	1	20	83	17	19	22

Table 2. ECFG inference results for XML documents.

the results to the corresponding DTDs (see Table 2 and Figure 6). Despite the essential growth in size and the initial number of nonterminals, the diversity of nonterminal contents does not increase much, and this results in the nonterminal merge behavior similar to separate files in the collections (see Figure 6.b).

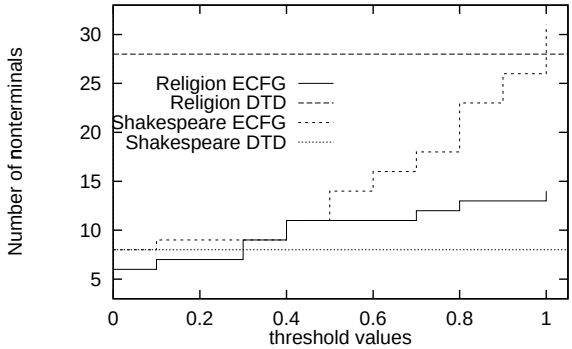

Fig. 5. Nonterminal merge for Religion and Shakespeare collections.

Next, we test the schema extraction algorithm when parameter n (used for n-grams in the content similarity) takes values 1,2 and 3. The merge results for `hamlet.xml` and `bom.xml` files are shown in Figure 6.a and Figure 6.b, respectively.

Finally, we report the results of content generalization for elements in one of the collections, the religious one. In Table 3 we group the outcomes obtained for th=1 in three parts. The first part shows the

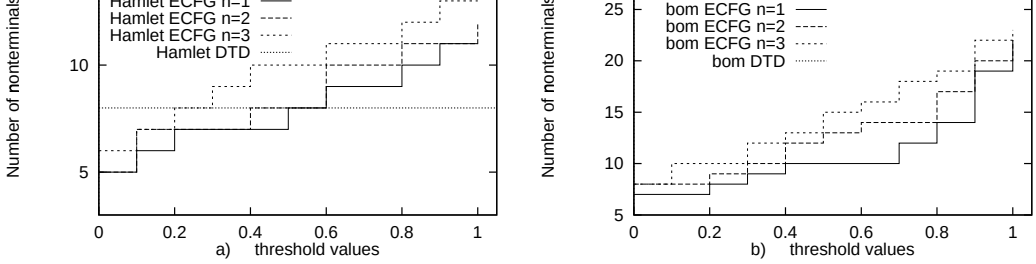

Fig. 6. Nonterminal merge for sample documents for $n = 1, 2, 3$: a) `hamlet.xml`, b) `bom.xml`.

simple range tightening for elements `subtitle` and `witlist`, etc. In the second group, simplification is detected for 9 elements such as `bktlong` and `bktshort` which are defined as complex ones in the DTD but appear as simple ones in all collection files (this actually explains the low number of nonterminals in inducted schema for the collection). In the third group, both range tightening and simplification are detected for elements `coverpg, titlepg, bookcoll, book, sura` and `chapter`.

Element	DTD definition	Tight range regular expression
subtitle	(p)+	p[1-3]
witlist	(witness)+	witness[3-8]
...		
bktlong	(#PCDATA — i)*	#PCDATA
bktlong	(#PCDATA — i)*	#PCDATA
...		
coverpg	(title|title2)+,(subtitle|p)*	title,title2,subtitle[2]
titlepg	(title|title2)+,(subtitle|p)*	title,title2,p[3]
bookcoll	(book—sura)+	book[3-39],(sura[1-3], book[1-3])[0-2]
book	bktlong,bktshort,epigraph?, bksum?,chapter+	bktlong,bktshort,bksum?,chapter[1-150]
sura	bktlong,bktshort,epigraph?, bksum?,v+	bktlong,bktshort,(epigraph—bksum),v[3-286]
chapter	chtitle,chstitle?,epigraph?, chsum?,(div+—v+)	chtitle,(chstitle—chsum)?,v[3-77]|div[22]

Table 3. Content generalization for elements of the religion collection.

6.1 Discussion

DTD style. One important discovery of conducted experiments is that the major part of XML documents available on the Web is designed in "DTD style", when *some elements play the role of complex types*. In the soccer data, the content of elements `ChLeague` and `UEFA` do have the same content; this fact is captured by the complex type *ListType*. Instead, with a "DTD-like" design, the XML document is likely to have an additional intermediate element, say `List`, which would play the same role as *ListType* type; the corresponding fragment of DTD definition being as follows:

```
<!ELEMENT ChLeague (List)>
<!ELEMENT UEFA (List)>
<!ELEMENT List (year result)*>
```

Deterministic model. Instantiation of deterministic regular expressions can bring surprising results when the resulting range regular expression is not longer deterministic. As an example, consider the regular example $(a+\ b*)*$ and its instantiation $r = (a[1:2]b[0:2])[4]$. Validation the input string $aabbaaab$ with r is non-deterministic, it requires lookahead 3. Therefore, the model of deterministic regular expression studied for infinite regular expressions in DTDs requires certain refinement in the case of range regular expressions.

Simple types extraction. An important extension of the schema extraction is the inference of simple types for elements with *structured patterns* in their values. Examples are zip and postal code, stock values, bibcode, etc. Identifying the minimally required datatypes (see Section 4) often provides a poor solution in such cases (often `string` datatype), as it is incapable to detect any structured patterns in the element values. If found, such structured patterns can be represented as derivation from elementary types in XML Schema language.

Reasonably required datatypes and range occurrences. The induction of minimally required datatypes and tight range occurrences works correctly, when the XML documents remain unchanged over time. Unfortunately, this assumption rarely holds on the Web, where the large part of real HTML/XML data undergo permanent changes. As analyses show (Sigmod Record archive is an example), more frequent changes are the addition of new elements at different levels. As consequence, the occurrence ranges appear to be most vulnerable to eventual changes. In the soccer example, the root element teams has 46 teams and clearly, the indicated range `team`[46] is inadequately tight.
Therefore, the schema extracted from a document collection should not be *too* tight, in order to avoid that *small* changes happening to data would not break the schema down. Inducing *reasonably require datatypes* for elements instead of minimally required ones may appear be more appropriate. Yet, the induction of such *soft schema* might require a statistical analysis of range distributions, in order to approximate possible changes and estimate reasonable occurrence ranges. Moreover, the result of statistical analysis can be accurate only in the case of sufficiently large set of examples. Otherwise, in the case of few samples, like one string for *teams* in the soccer example, the statistical analysis is not applicable and other solutions are to be proposed.

7 Conclusion

We have developed a novel schema extraction from XML documents based on the powerful model of extended context-free grammars. To our knowledge, this is the first attempt to induce XML schema that unifies the expressive power of ECFGs and the determinism requirement. We have identified three important components of the extraction algorithm, namely, the grammar induction itself, content generalization and tight datatype identification. While the second problem appears also in the DTD extraction, the first and third ones are relevant to the new schema model, we have proposed sophisticated solutions for each of them. We have tested the proposed algorithm on the set of real XML documents and validated the ECFG induction method based in the content and context similarity of nonterminals.

References

1. H. Ahonen, H. Mannila, and E. Nikunen. Forming Grammars for Structured Documents: an Application of Grammatical Inference. In Rafael C. Carrasco and Jose Oncina, editors, *Proceedings of the Second International ICGI Colloqium on Grammatical Inference and Applications*, volume 862 of *LNAI*, pages 153–167, Berlin, September 1994. Springer Verlag.
2. H. Ahonen, H. Mannila, and E. Nikunen. Generating Grammars for SGML Tagged Texts Lacking DTD. In *M. Murata and H. Gallaire (eds.), Proc. Workshop on Principles of Document Processing (PODP)*, 1994.
3. Helena Ahonen. Automatic Generation of SGML Content Models. Proc. Sixth Intern. Conf. on Electronic Publishing, Document Manipulation and Typography, 1996. 195-206.
4. D. Angluin. Computational Learning Theory: Survey and Selected Bibliography. In N. Alon, editor, *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 351–369, Victoria, B.C., Canada, May 1992. ACM Press.

5. P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes. http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/.
6. A. Bruggemann-Klein and D. Wood. One-Unambiguous Regular Languages. *Information and Computation*, 140,142(2):229–253, 182–206, 1998.
7. Moh C.-H., Lim E.-P., and Ng W.-K. Re-engineering Structures from Web Documents. In *ACM Digital Libraries 2000, San Antonio, Texas, USA*, pages 67–76, June 2000.
8. D.C Fallside. XML Schema Part 0: Primer. W3C Candidate Recommendation. http://www.w3.org/TR/xmlschema-0/.
9. C. Faloutsos and D. Oard. A Survey of Information Retrieval and Filtering Methods. Technical report, University of Maryland, College Park, MD, 1996.
10. D. Florescu and D. Kossmann. Storing and Querying XML Data Using an RDBMS. *IEEE Data Engineering Bulletin*, 22(3), 1999.
11. Fred. The SGML Grammar Builder. http://www.oclc.org/fred/.
12. Minos N. Garofalakis, Aristides Gionis, Rajeev Rastogi, S. Seshadri, and Kyuseok Shim. XTRACT: A System for Extracting Document Type Descriptors from XML Documents. In *Proc. ACM SIGMOD, Dallas, Texas, USA*, pages 165–176. ACM, 2000.
13. Roy Goldman and Jennifer Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proc. 23rd Intern. Conf. on Very Large Data Bases*, pages 436–445, 1997.
14. et al. H. S. Thompson. XML Schema Part 1: Structures. http://www.w3.org/TR/2000/CR-xmlschema-1-20001024/.
15. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, N. Reading, MA, 1980.
16. R. Jelliffe. Schematron. http://www.ascc.net/xml/resource/schematron/, May 2000.
17. Ke Wang and Huiqing Liu. Discovering Structured Association of Semistructured Data. *IEEE Trans. Knowledge and Data Engineering*, 12(3):353–371, 2000.
18. Dong-Won Lee and Wesley W. Chu. Comparative Analysis of Six XML Schema Languages. *SIGMOD Records*, 29(3), September 2000.
19. Dongwon Lee, Murali Mani, and Makoto Murata. Reasoning about XML Schema Languages using Formal Language Theory. Technical report, IBM Almaden Research Center, RJ 10197, Log 95071, 2000.
20. M. I. Schwatzbach N. Klarlund, A. Moller. DSD: A Schema Language for XML. Proc. 3rd ACM Workshop on Formal Methods in Software Practice, 2000.
21. Yannis Papakonstantinou and Victor Vianu. DTD Inference for Views of XML Data. In *Proc. of 19 ACM Symposium on Principles of Database Systems (PODS), Dallas, Texas, USA*, pages 35–46, 2000.
22. D. Quass, J. Widom, R. Goldman, K. Haas, Q. Luo, J. McHugh, S. Nestorov, A. Rajaraman, and H. Rivero. LORE: A Lightweight Object Repository for Semistructured Data. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(2):549, 1996.
23. D. Raymond and D. Wood. The User's Guide to Grail. University of Waterloo, Waterloo, Canada, 1995.
24. A. Sahuguet. Everything you ever wanted to know about dtds, but were afraid to ask. In *Proc. 3rd Int'l Workshop on the Web and Databases (WebDB), Dallas, TX*, 2000.
25. Y. Sakakibara. Learning Context-Free Grammars from Structural Data. In *Proc. 1988 Workshop on Computational Learning Theory*, pages 330–344. Morgan Kuafmann Publishers, 1988.
26. Yasubumi Sakakibara. Efficient Learning of Context-Free Grammars from Positive Structural Examples. *Information and Computation*, 97(1):23–60, March 1992.
27. J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proc. Intern. Conf. on Very Large Databases, Edinburgh, Scotland*, 1999.
28. T. Bray, J. Paoli, and C.M. Sperberg-McQueen. XML Extensible Markup Language 1.0. W3C Recommendation, http://www.w3.org/TR/1998/REC-xml-19980210.html.
29. D. Wood. Standard Generalized Markup Language: Mathematical and philosophical issues. *Lecture Notes in Computer Science*, 1000:344–365, 1995.
30. Y. Sakakibara. Recent Advances of Grammatical Inference. *Theoretical Computer Science*, 185(1):15–45, October 1997.