Schema Extraction from XML Data: A Grammatical Inference Approach

Boris Chidlovskii
Xerox Research Centre Europe, Grenoble Laboratory, France
6, Chemin de Maupertuis, F–38240 Meylan, 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 results of some tests on real XML collections.

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 [20]. 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 [6, 19].

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 [5, 11, 15]. 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 [13]. 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 [3, 5, 10].

Problem statement. Despite the importance of schema information, schema definitions are not obligatory in XML. This raises the *problem of extracting the schematic information* from XML documents and collections. 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 contradict each other, finding out an optimal tradeoff is a challenging task.

Automatic schema extraction can bring several important advantages. 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 [17]. Second, the automatic schema extraction can be beneficial for in mediator systems that process *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 (constraining or loosing) the existing schema patterns and finding new ones.

Our contribution. We adopt the XML schema formalism based on *extended context-free grammars* (ECFG) already used in [16, 21]. 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.

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 difficult 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 (limited) set of regular grammars.

We identify three important problems in the schema extraction: (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 due 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 [18]. For the third problem (datatypes constraining), we develop an algorithm based on the subsumption relationship among elementary datatypes in XML Schema language [3]. Finally, for the second problem (content generalization) we propose a solution alternative to those used for DTD extraction [4, 8]; it copes with the determinism requirement and the occurrence constraints in XML Schema language.

Both DTDs and XML Schema require an easy validation of XML data against corresponding schema [10, 20]. Such an ease is defined as "determinism" and closely corresponds to the notion of unambiguity in the formal language theory.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.

We align the schema extraction method with the XML Schema language. On one hand, it automatically treats requirements imposed by XML Schema language, such as determinism, and on other hand, it allows the user to tune the induced schema by varying a generalization parameter as a way to an optimal tradeoff between the schema tightness and compactness.

State of art. XML data model is a modification of the semi-structured data model. For semi-structured data, schema may have the form of Data Guides [9] that represent a condensed graph-like representation of semi-structured data. A Data Guide is obtained from the semi-structured data by removing element duplications and repeating paths. An alternative approach is in finding the most frequent structural patterns in a semi-structured database [12].

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 [14, 16]. Moreover, DTDs are limited from the point of view of XML querying, as no *tight* DTDs can be induced even for view queries [16]. The possible solution to the tightness problem requires the extension of DTDs with the sub-typing and specialization, it makes the resulting schema model very close to the XML Schema language [16].

Five novel XML schema languages have been recently compared to DTDs in [13], that classified all important features for defining XML schema, such as simple and complex types, elementary datatypes, constraints for elements/attribute values and appearance, inheritance. While DTDs appear to have the weakest expressive power, the most powerful languages are XML Schema [5], Schematron [11] and DSD languages [15].

Automatic DTD extraction systems have been first designed for SGML [1, 7], some of them have been later extended to XML [4]. An advanced algorithm for extracting DTDs from XML documents is proposed in the XTRACT system [8]. 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 candidate.

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 [18]. Second, we use regular grammar inference methods [2] in order to generalize content strings into regular expressions.

The remainder of the paper is organized as follows. Section 2 introduces the ECFG schema formalism and Section 3 explains the schema induction algorithm on an example. In Section 4, we analyze the the nonterminal merge during an ECFG inference, while we leave two other problems, constraining datatypes for simple elements and generalization of content strings, for the full version. Section 5 and 6 report some inference experiments and conclude the paper.

2 Extended context-free grammars as XML schema

As example, we consider an XML data about European soccer clubs. 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>...</ChLeague>// 5 successes in total
          <UEFA><year>1995</year><result>final</result>...</UEFA>// 7 successes in total
       </team>
       <team><name>Manchester United</name>
          <ChLeague><year>1999</year><result>winner</result>...</ChLeague>// 4 successes in total
       </team>
       <team><name>Hertha Berlin</name></team>
       ...  // 46 teams in total
<teams>
```

XML Schema language.XML Schema language [3, 5, 10] offers a large 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: (1) *elementary datatypes* for simple elements (with no sub-elements), (2) *complex types* for complex elements (with sub-elements), (3) *sequence* and *choice* groups of elements, (4) *occurrence constraints* for elements and groups (`MaxOccurs`, `MinOccurs`.).

Here 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` 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>
```

2.1 Extended context-free grammars

We model XML schema as range extended context-free grammars. ECFGs have been already used as XML schema model in [16, 21]. We adopt the model and extend it with range

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.

occurrences 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 [21], 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 [6, 19].

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' are a pair of opening/closing tags, $t, t' \in T$ and $B \in N \cup D$. The notion of term is to capture the well-formedness of XML [20]. Without loss of information, we abbreviate a term $t\ B\ t'$ as $t : B$.

XML Schema and ECFGs. The ECFG-based schema model addresses the previously chosen subset of features in the XML Schema language in the 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.

Example 1 *Consider the XML Schema definition for the soccer XML data. 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 [8].

2.2 Determinism and extended context free grammars

By both XML DTDs and XML Schema requirements [10, 20], 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"[10].

It turns out that the determinism essentially constrains the power of ECFG model, as 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 that 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. Therefore, we distinguish between *vertical* and *horizontal* determinism, as they address the vertical and horizontal navigation through an XML data tree [14]. Here we consider the vertical determinism, and the horizontal determinism is discussed in the full version.

The *vertical determinism* requires that at any complex element, the rule for any sub-element should be determined with one-token lookahead. Formally, we say that two terms $t : A$ and $t' : A'$ are *ambiguous* if $t = t'$ but $A \neq A'$. Ambiguous terms in productions make difficult parsing and validating XML documents [21]. For example, in the production $A = t : A' \mid t : A''$, the correct choice between the first and second terms will require lookahead 2 or more 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.*

2.3 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 [18]. 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 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.

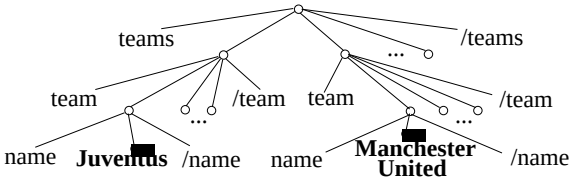

Figure 1: Example XML data as structural example.

3 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**. 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:

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

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

Start → `(team:`A_1`)[46]` *Step 1.2.b*
A_1 → `name:`**Any** `ChLeague:`A_{47} `UEFA:`A_{47}`| name:`**Any** `ChLeague:`A_{47}`|...`
A_{47} → `(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:

Start → `(team:`A_1`)[46]` *Step 1.4*
A_1 → `name:`**String** `(ChLeague:`A_4`)[0:1] (UEFA:`A_4`)[0:1]`
A_4 → `(year:`**UnsignedShort** `result:`**String**`)[0:8]`

Comparing the resulting ECFG to the initial one in Section 2, one can observe the minimal difference concerning 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 steps, the merge of nonterminals and constraining datatypes for simple elements are specific to the ECFG schema model only. In the following section we discuss the nonterminal merge; the datatype constraining and generalizing content strings in regular expressions are presented in the full version.

4 Nonterminal merge

Assume the initial set N of nonterminals is built from a set of structured examples. In the context-free grammar inference from structural examples [18], the induction is conducted by merging nonterminals with *equivalent content* and *equivalent context* as follows:

1. **Content equivalence:** $A \to \alpha$, $B \to \alpha$ in δ implies that $A = B$,
2. **Context equivalence:** $A \to \alpha B \beta$, $A \to \alpha C \beta$ 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 equivalence conditions appear to be too strong; they fail to identify and merge nonterminals whose right parts are two different instances of one regular expression (nonterminals A_{47} and A_{48} at Step 1.2.a). Therefore, we replace the equivalence conditions in (1) and (2) with weaker ones.

For the determinism requirement, we implement the condition established in Section 2.2 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:** $A \to \ \alpha\ t{:}B\ \beta\ t{:}C\ \gamma$ implies that $B = C$.

While the merge of productions with ambiguous terms is imposed by the determinism requirement, the merge of nonterminals with similar content may remain optional and requires some similarity metric.

Consider again the soccer example and denote elements `year`, `result` and datatype `string` as y, r and s, respectively. Then non-terminals $A_{47} \to (y{:}s\ r{:}s)[7]$ and $A_{48} \to (y{:}s\ r{:}s)[5]$, are likely that their right parts are instances of the range regular expression $(y{:}s\ r{:}s)[l{:}r]$ and therefore A_{47} and A_{48} can be merged. Below we propose the rule 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. 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, $n = 1, 2, 3, \ldots$. 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 $\mathcal{M}(c_1, c_2) \geq th$, where $0 \leq th \leq 1$ is a threshold value.

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 $A \to \alpha$ and $B \to \beta$ are such that $\alpha = c_1|c_2|...$ and $\beta = c'_1|c'_2|...$, then

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

So, we can rewrite the strong content equivalence condition (1) with a weaker one:

4. **Content similarity:** $A \to \alpha$, $B \to \beta$ in Σ, such that $\mathrm{M}(\alpha, \beta) \geq th$, implies that $A = B$.

5 Some tests

We have tested the schema extraction algorithm on three different XML collections (with their DTDs), namely, Shakespeare plays and astronomy files available at `http://www.goxml.org/` and the Sigmod Record archive available at `http://www.dia.uniroma3.it/Araneus/Sigmod/`. Here we report some results of the nonterminal merge (see Section 4), with the nonterminal merge threshold th varying in the range [0,1], to see how well it wraps up the initial (usually enormous) number of nonterminals.

Figure 2 shows the performance of the nonterminal merge for three documents, `hamlet.xml` from Shakespeare collection, `1021A.xml` from astronomy collection and `sigmod.xml`. The figure plots the final 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 nonterminal merge, imposed by the vertical determinism requirement. As the plotting shows, the vertical determinism provides a satisfactory reduction of nonterminal numbers, comparable to the size of corresponding DTD.

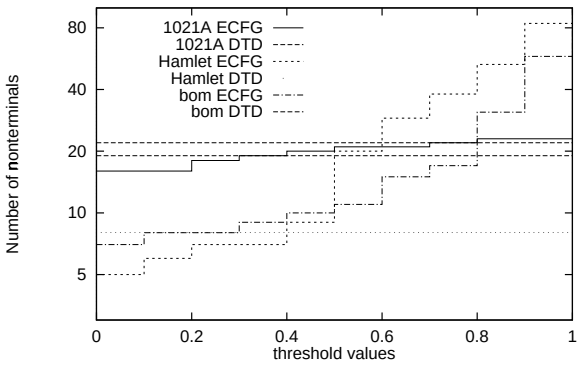

Figure 2: Nonterminal merge for three sample documents.

Small values of threshold th for the content similarity make easier the merge of remaining nonterminals and can further reduce their number. In the case of `sigmod.xml` file, however, the vertical determinism has reached the maximal reduction and no additional merge is possible with the content similarity, whatever the th value is.

6 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.

The generalization of sample XML documents into ECFG schema is driven by the determinism requirement and a given threshold parameter th for the content similarity. We note that it does not address the problem of optimal value of th, since the *selection of optimal XML schema* is a subject of our further study. On one hand, the Minimal Description Length principle already used for the DTD extraction [8] can help to choose between different schema candidates. It should be however properly adopted, as the extracted ECFG schema are finite languages, and the language size should be considered at least as important as the description length. On the other hand, we may consider some alternative criteria for the schema optimality. These criteria can be simple ones, like limiting the number of nonterminals/complex types, or more sophisticated ones, like providing the optimal execution for a given set of XPath queries, etc.

References

[1] Helena Ahonen. Automatic Generation of SGML Content Models. Proc. Sixth Intern. Conf. on Electronic Publishing, Document Manipulation and Typography, 1996. 195-206.

[2] 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.

[3] P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes. http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/.

[4] 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.

[5] D.C Fallside. XML Schema Part 0: Primer. W3C Candidate Recommendation. http://www.w3.org/TR/xmlschema-0/.

[6] D. Florescu and D. Kossmann. Storing and Querying XML Data Using an RDBMS. *IEEE Data Engineering Bulletin*, 22(3), 1999.

[7] Fred. The SGML Grammar Builder. http://www.oclc.org/fred/.

[8] 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.

[9] 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.

[10] et al. H. S. Thompson. XML Schema Part 1: Structures. http://www.w3.org/TR/2000/CR-xmlschema-1-20001024/.

[11] R. Jelliffe. Schematron. http://www.ascc.net/xml/resource/schematron/, May 2000.

[12] Ke Wang and Huiqing Liu. Discovering Structured Association of Semistructured Data. *IEEE Trans. Knowledge and Data Engineering*, 12(3):353–371, 2000.

[13] Dong-Won Lee and Wesley W. Chu. Comparative Analysis of Six XML Schema Languages. *SIGMOD Records*, 29(3), September 2000.

[14] 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.

[15] M. I. Schwatzbach N. Klarlund, A. Moller. DSD: A Schema Language for XML. Proc. 3rd ACM Workshop on Formal Methods in Software Practice, 2000.

[16] 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.

[17] 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.

[18] Yasubumi Sakakibara. Efficient Learning of Context-Free Grammars from Positive Structural Examples. *Information and Computation*, 97(1):23–60, March 1992.

[19] 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.

[20] 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.

[21] D. Wood. Standard Generalized Markup Language: Mathematical and philosophical issues. *Lecture Notes in Computer Science*, 1000:344–365, 1995.