On the Complexity of Standard and Specialized DTD Parsing

Ralf Behrens

Institut für Informationssysteme
Osterweide 8, 23562 Lübeck
Medizinische Universität zu Lübeck, Germany
eMail: behrens@ifis.mu-luebeck.de

Abstract. We investigate an extension of standard DTDs named *s-xschemes* yielding more expressive power. We show that *s-xschemes* which are also known as specialized DTDs have more convenient complexity properties than context-free languages. We invent the notions of *properness* and *1-unambiguity* for a close characterization of *s-xschemes*, whereby "1-unambiguity" is a restriction concerning expressive power while "properness" is not. With these conditions we show an efficient algorithm for parsing in linear time and space. We propose a concrete syntax and provide an implementation inside a parsing system. [1]

1 Introduction

XML DTDs serve as a grammar-like syntax for specifying common structures for classes of XML documents. Since DTDs have widely agreed shortcomings [1], there are several approaches to overcome them. [12, 13] propose an approach which describes schematic information by XML documents themselves. Thereby they can define more precise structure restrictions, types for attributes etc. However, parsing of such schemas is rather complex. Alternatively we proposed so called *s-xschemes* [2] which are similar to the generalized DTDs of [11]. In contrast to [11] we use these extensions not only to characterize views and query results, but also to define XML documents' schemas. Our *s-xschemes* are especially important for more strict schema definition and grammar based metadata extraction as shown in [2].

2 Parsing Methods for XML Documents

In this paper we use the notions of [2], namely *xrules*, *xschemes* and *s-xschemes*. While an *xrule* is the abstraction of an `ELEMENT` statement in XML, which ties an element name together with a content model, an *xscheme* d specifies the structural part of a DTD. An *s-xscheme* d_s is an extension of an *xscheme*, yielding more expressive power by the use of a homomorphic function. The *s-xschemes* are especially important for schema union and finite recursion. Further more they are used for handling document structures on different levels of abstraction and for extraction of metadata. The combination of an element name together with a regular expression is called *xrule*, denoted $\langle e : type(e) \rangle$. An *xscheme* is a set of *xrules* together with a unique start element. For *s-xschemes* we use homomorphic functions. A homomorphic function can be applied to element names, as well as words, *xtrees* and sets, and will be called h here. The expression $h(type(e))$ provides a regular expression that is obtained by replacing each element name by its homomorphic image. $L(d)$, $L(d_s)$ denote the textual language defined by the *xscheme* d resp. *s-xscheme* d_s. Given an *s-xscheme*, the *simplified xschemes* are those obtained by replacing each element of an *xrule* by its homomorphic image. Since an *s-xscheme* can have several start symbols, multiple simplified *xschemes* are possible.

[1] Published in 12. GI-Workshop "Grundlagen von Datenbanken", pages 6-10

Figure 1 shows a DTD describing a bookstore with novels and journals. The DTD is abstracted by an *xscheme*.

```
<!ELEMENT bookstore   (novel, journal)>
<!ELEMENT novel       exemplar*>
<!ELEMENT journal     exemplar*>
<!ELEMENT exemplar    title, author?>
<!ATTLIST exemplar    id ID, publisher IDREF,
                      status="published|unpublished">
<!ELEMENT title       (#PCDATA)>
<!ELEMENT author      (#PCDATA)>
```

bookstore	: novel, journal
novel	: exemplar*
journal	: exemplar*
exemplar	: title, author?

Figure 1. An example of an XML-DTD and its *xscheme*

In the rest of the paper we will use the notion of *xscheme* resp. *s-xscheme* when the structural abstraction is considered, and the notion of (specialized) DTD, when we talk about the concrete syntax. As a first demonstration we want to define a specialized DTD that, opposed to the former example, only allows novels with title followed by author, resp. journals with a title:

```
<!ELEMENT bookstore    novel,journal>          <bookstore> <novel>
<!ELEMENT novel        n_exemplar*>             <exemplar>
<!ELEMENT journal      j_exemplar*>              <title>Winter</title><author>Meyer</author>
<!ELEMENT n_exemplar   (title, author)>         </exemplar>
<!ELEMENT j_exemplar   title>                   </novel>
<!ELEMENT title        (#PCDATA)>               <journal>
<!ELEMENT author       (#PCDATA)>                <exemplar><title>Focus</title></exemplar>
<!MAPPING exemplar     n_exemplar, j_exemplar>  </journal> </bookstore>
```

Figure 2. An example of a specialized DTD and a valid XML document (Attributes are omitted)

As we can see in Figure 2, we invent a special statement `MAPPING` that ties different versions to an element name. These versions are called *candidate archetypes*, what is different to the notion of *archetype*.

Definition 1 (Archetype of an Element). Let t be an *xtree* valid in an *s-xscheme* d_s, t' be any other *xtree* such that $t' \in h^{-1}(t)$. Let n, n' be two corresponding nodes in t resp. t', e, e' are the elements corresponding to n resp. n'. Now e' is called an *archetype* of e with regard to n.

Consequently we call an *xtree* which is built by replacing an element name with one of its archetypes an *archetype xtree*. In contrast to an archetype the *candidate archetype* of an element e is every element e' with $h(e') = e$. Clearly an archetype element is also a candidate one, but this does not have to hold vice versa.

Parsing of an *xtree* with regard to an *s-xscheme* could clearly be done in the following manner. For an *xtree* we build all possible archetype trees, i. e. their homomorphic image is the origin tree. Now we parse each of these trees by a usual XML parser, which lets us find an archetype tree. Unfortunately this would run into exponential time complexity, and is therefore not practical. Thus we will show a more efficient way for *s-xscheme* parsing here.

Definition 2 (Proper *s-xscheme*). An *s-xscheme* d_s is called *proper*, iff for every *xtree* t in $L(d_s)$ the condition $|h^{-1}(t) \cap h^{-1}(L(d_s))| = 1$ holds.

Properness ensures that there is always exactly one archetype element for a given node of an *xtree*.

Example 3.

```
<!ELEMENT bookstore    (novel + journal)*>
<!ELEMENT novel        volume>
<!ELEMENT journal      volume>
<!ELEMENT volume       (#PCDATA)>
<!MAPPING exemplar     novel, journal>
```

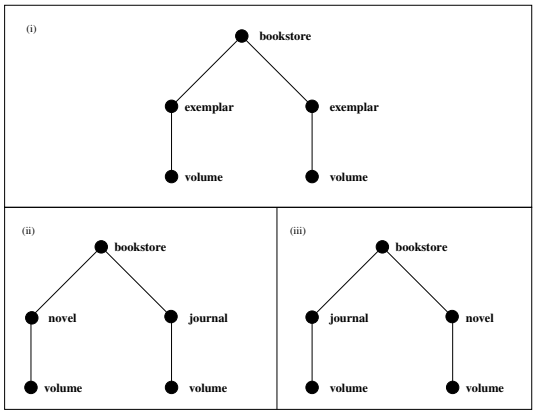

Figure 3. Unproper specialized DTD

We take a look at the unproper specialized DTD in Figure 3. We build an *xtree* with different archetypes here. Using the homomorphic function, the *xtree*s (ii) and (iii) are mapped onto the same *xtree* (i).

We can show that *properness* does not restrict *s-xschemes*' expressive power:

Theorem 4 (Existence of proper *s-xscheme*). *For every s-xscheme d_s we can find a proper s-xscheme $d_{s'}$ with $L(d_s) = L(d_{s'})$.*

Proof:

According to [7, 2] for every language described by a specialized DTD we can build a regular tree automaton accepting the same language. Every nondeterministic tree automata can be transformed into a deterministic one by power set creation over its states. In a similar manner the archetypes of an *s-xscheme* can be refined, such that the *s-xscheme* gets proper. □

Thus from now on we expect an *s-xscheme* to be proper, which implies that the archetype of an *xtree* is uniquely defined within $h^{-1}(L(d_s))$.

Example 5. The specialized DTD of Example 3 has a proper version as follows:

```
<!ELEMENT bookstore      novel_journal*>
<!ELEMENT novel_journal  volume>
<!ELEMENT volume         (#PCDATA)>
<!MAPPING exemplar       novel_journal>
```

XML content models are unambiguous. However, the notion of unambiguity in XML differs significantly from that of formal language theory. Therefore we will first define and distinguish these two concepts. To indicate different positions of the same symbol in a regular expression, symbols can be marked with subscripts. For example, $(a_1 + b_1)^* a_2 (a_3 b_2)^*$ and

$(a_4 + b_2)^* a_1 (a_5 b_1)^*$ are both markings of the expression $(a + b)^* a (ab)^*$. For each expression r over Σ, a marking is denoted by r'. Unambiguity in the sense of [5] means that every child of a node n with $name(n) = e$, which is generated by an *xrule* $\langle e : type(e) \rangle$, can be assigned to exactly one position in $type(e)$. For example, consider the expression $(a + b)^* aa^*$. If we mark different positions of the same symbol with subscripts, we get $(a_1 + b_1)^* a_2 a_3^*$. Now three subscripted words, namely $a_1 a_2 a_3$, $a_1 a_2 a_3$, and $a_2 a_3 a_3$ stand for the same word aaa. Thus $(a + b)^* aa^*$ is ambiguous. However, there is an unambiguous expression that describes the same language as $(a + b)^* aa^*$, namely $(a + b)^* a$.

Proposition 6 (Unambiguity of a Regular Expression). *For every regular language R we can find an unambiguous regular expression r, such that $L(r) = R$ [5].*

In contrast to [5], unambiguity as defined in XML is a one-symbol-lookahead unambiguity. In [6] this unambiguity is called "1-unambiguity".

Definition 7 (1-Unambiguity). An expression r is 1-unambiguous if, and only if for all words u, v, w over Σ and all symbols x, y in Σ, the conditions $(uxv)', (uyw)' \in L(r')$ and $x' \neq y'$ imply $x \neq y$. A regular language is 1-unambiguous if it is denoted by some 1-unambiguous expression.

In other words, for each word w described by a 1-unambiguous expression r, there is exactly one marked word w' that can be build symbol by symbol without looking ahead. Consider the expression $(a + b)^* a$ marked as $(a_1 + b_1)^* a_2$. In the word baa, after we match symbol b at position b_1, we could not decide whether we should match the subsequent a in the word with position a_1 or a_2 without looking beyond the current symbol a in the word. Therefore, although $(a + b)^* a$ is unambiguous, it is not 1-unambiguous. However, $b^*(a(b^* a))^*$ is a 1-unambiguous expression that denotes the same language as $(a + b)^* a$. Consequently an 1-unambiguous *xscheme* is one, where each regular expression for itself is 1-unambiguous.

Every XML document can be parsed with regard to its structure in linear time and space. This is due to the fact that every child sequence which is described by a 1-unambiguous regular language can be parsed in linear time and space. Thus, regarding an *xtree* we have to test for each node, whether its child sequence is valid in the corresponding *xrule*. This had to be done k times (depending on the number of inner nodes) where for each test t_i child nodes are involved such that $\Sigma_{i=1}^{k} t_i = n$. The reason why this way of parsing does not work for *s-xschemes* is its eventually ancestor sensitivity. Parsing an *xtree* top-down, we do not know in general (i. e. without looking several nodes ahead) the archetype of each node.

Definition 8 (1-Unambiguity of an *s-xscheme*). An *s-xscheme* is called *1-unambiguous*, if every regular expression $h(r_e)$ of an *xrule* is 1-unambiguous.

If we carefully look at the condition of 1-unambiguity we can see that it restricts the expression power of *s-xschemes*. Cases using the $*, ?$ operators can be not be transformed into 1-unambiguous forms in general.

However the condition of 1-unambiguity enables us to parse documents without any look-ahead in linear time and space. This gives us the opportunity to enhance a conventional XML parser to capture our XML extension.

Lemma 9 (Parsing of *xtree*s). *Every XML document can be parsed with regard to an 1-unambiguous, proper s-xscheme in linear time and space.*

Algorithm[*s-xscheme*-Parsing of an *xtree*]

function *parse*(1-unamb. proper *s-xscheme* d_s, *xtree* t) **returns** archetype *xtree*
begin
Begin at the root n of t
Let n be labeled e and have a child label sequence $e_1 \dots e_k$
Find an *xrule* $\langle e : r \rangle$ such that $e_1 \dots e_k \in L(h(r))$
if no *xrule* exist **quit**
Rename each node by its archetype
for each subtree t_i (which is no leaf) rooted by a child node **do** *parse*(d_s, t_i)
return t
end

We have implemented a parser that supports our results. The parser is based on Javasoft's *Project X*-Parser [8] and it is completely designed in Java.

3 Conclusion

The main contribution of this paper is the development of sufficient conditions for specialized DTDs in order to enable parsing of documents in linear time and space. We presented an efficient parsing algorithm. Our results are implemented into a concrete parsing system [9]. This parser will be used in our media archive MONTANA [10, 4, 3].

Acknowledgement: The present author would like to thank Volker Linnemann for fruitful discussions on the present topic.

References

1. C. Beeri and T. Milo. Schemas for integration and translation of structured and semi-structured data. In *International Conference on Database Theory (ICDT)*, pages 296–313, 1999.
2. R. Behrens. A grammar based model for XML schema integration. In *British National Conference on Databases (BNCOD)*, pages 172–190, 2000.
3. R. Behrens. MONTANA: Towards a web-based infrastructure to improve lecture and research in a university environment. In *2nd International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems*, pages 58–66, San Jose, 2000.
4. R. Behrens, C. Lecon, V. Linnemann, and O. Schmitt. MONTANA: Ein Medienarchiv für die Lehre. Technical report, Institut für Informationssysteme, Med. Univ. Lübeck, September 1998. Schriftenreihe der Institute für Informatik/Mathematik, A-98-19; available at `http://www.ifis.mu-luebeck.de/public/`.
5. R. Book, S. Even, S. Greibach, and G. Ott. Ambiguity in graphs and expressions. *IEEE Transactions on Computers*, 20:149–153, 1971.
6. A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 140(2):229–253, 1998.
7. A. Brüggemann-Klein and D. Wood. Regular tree languages over non-ranked alphabets. Working paper, 1998.
8. Javasoft. JAXP Homepage. `http://www.javasoft.com/xml/`.
9. M. Kreutzberger. Konzeption und Implementierung eines Parsers auf Basis einer XML Erweiterung. Medizinische Universität Lübeck, 2000. Studienarbeit.
10. Homepage of the digital media archive MONTANA. `http://passat.mesh.de:4040`.
11. Y. Papakonstantinou and P. Velikhov. Enhancing semistructured data mediators with document type definitions. In *International Conference on Data Engineering (ICDE)*, pages 136–145, 1999.
12. XML Schema Part 1: Structures. W3C Working Draft, `http://www.w3.org/TR/xmlschema-1/`, 1999.
13. XML Schema Part 2: Datatypes. W3C Working Draft, `http://www.w3.org/TR/xmlschema-2/`, 1999.