Formal Language Foundations
and Schema Languages

Stefan Tittel

University of Dortmund

Overview

This paper deals with the expression power and decidability of XML schema languages. In the first section we will analyze Document Type Definitions (DTDs) by transforming them to string grammars and taking a closer look at the resulting language classes. In the second section *one-unambiguity*—a property of language classes that will proof itself very useful for further analysis—is introduced and methods are given, which can be used to determine if a certain regular expression or regular language is in fact one-unambiguous. Finally in the third section of this paper we will compare XML schema languages using formal language theory.

1 XML Languages and Grammars

1.1 Introduction and Basics

XML is a general-purpose markup language widely in use. However the syntactical constraints imposed by XML itself are too general to be sufficient for most applications. To further restrict the set of valid documents, XML schema languages are used to define the relative positions of pairs of corresponding tags and the kind of content that may occur between those tags. Document Type Definitions (DTDs) are a schema language native to the XML specification. We will transform DTDs to string grammars in order to analyze the language class generated by DTDs. To learn which languages can be expressed by DTDs can be highly relevant for practical applications; just consider a situation in which somebody plans to model his constraints using a DTD and wants to know in advance if this is actually possible.

To do such a transformation we will first need to define what kind of grammar we consider to be a DTD grammar (we will call those grammars *XML grammars* to conform with existing literature). Naturally the set of terminal symbols has

to be the set of opening and closing tags. Furthermore there shall be exactly one non-terminal symbol for each sort of tags and a start symbol being an element of the set of non-terminal symbols. For each non-terminal X_a (corresponding to the sort of a tags) there shall be a production rule whose right-hand side denotes a regular expression—which denotes what other sorts of tags may occur inside `<a></a>`—enclosed by an opening and a closing a tag. Note that we ignore empty tags and attributes at this point for reasons of simplicity and only consider reduced grammars.

We are going to need some further definitions. The language of *Dyck primes* (denoted by D or D_A) is the language of properly tag-parenthesized words over an alphabet of opening and closing tags (denoted by A and $\overline{A}$). It is not an XML language if the alphabet consists of more than one tag sort (just consider the alphabet $\{a, \overline{a}, b, \overline{b}\}$ and the words $a\overline{a}$ and $b\overline{b}$, which cannot both be generatable from the same XML grammar). However the subset of D_A in which each word starts with some tag a is an XML language and denoted by D_a. Furthermore we understand as set of *contexts* in L of a word w (denoted by $C_L(w)$) the set of pairs of words (x, y) such that $xyw \in L$.

1.2 Characterization

1.2.1 Conditions for a Language to Be XML

As has been motivated at the beginning, we want to get hold of conditions with which we can decide if a given language can be modelled by DTDs, i.e. if a given language is an XML language. The first condition is rather obvious: The given language must be a subset of D_a for some sort of tags a, i.e. all words of the language must be properly tag-parenthesized and start with the same tag (and because they are properly tag-parenthesized they will also end with a closing tag corresponding to that tag).

Let L be a language and $F(L)$ be the set of factors (i.e. all partial words) of L. In this case the set $F_a(L)$ is defined as $D_a \cap F(L)$. $F_a(L)$ therefore is the set of all factors of L which also are properly tag-parenthesized and start with an a tag. In an XML language, the Dyck primes that are allowed between a certain opening and closing tag, are independent of what occurs left to the opening and right to the closing tag. More formally this means that $C_L(w) = C_L(w')$ for all $w, w' \in F_a(L)$ and $a \in A$ (A being the set of opening tags). This is our second condition.

If w is a Dyck prime in D_a it can be uniquely factorized as $au_{a_1} u_{a_2} \cdots u_{a_n} \overline{a}$ with $u_{a_i} \in D_{a_i}$ for $i = 1, \ldots, n$. Then $a_1 a_2 \cdots a_n \in A^*$ is called the *trace* of the word w. The set of all traces of words in $F_a(L)$ is called the *surface* of a in L and is denoted by $S_a(L)$. For an XML language this set has to be regular, which is our third condition. We summarize these three conditions in the following theorem.

Theorem 1.1 *A language L over $A \cup \overline{A}$ is an XML language if and only if the following three conditions hold true:*

1. $L \subset D_\alpha$ *for some* $\alpha \in A$,

2. $C_L(w) = C_L(w')$ *for all* $a \in A$ *and* $w, w' \in F_a(L)$,

3. $S_a(L)$ *is a regular set for all* $a \in A$. $\square$

1.2.2 Closure Properties, Decidability and Further Results

Because of space constraints we will only have a closer look at the closure properties of XML languages regarding union and difference and give the other results right away:

- XML languages are closed under intersection.

- For each XML language L there is exactly one reduced XML grammar generating L if variable names and entities are ignored.

- It is decidable if an XML language L is included in or equal to another XML language M.

- It is also decidable if a regular language $L \subset D_A$ is an XML language.

- It is however undecidable if a context-free language is an XML language.

Regarding the closure under union consider the languages $L = D^*_{\{a,b\}}$, $M = D^*_{\{a,d\}}$, and $H = \{cL\overline{c}\} \cup \{cM\overline{c}\}$. As we observed in 1.1 the language of Dyck primes enclosed by a certain sort of tags is an XML language, therefore $cL\overline{c}$ and $cM\overline{c}$ both have to be XML languages and H as the union of those two languages also has to be an XML language if XML languages are closed under union. But we will show that this is not the case: Obviously the words $cab\overline{b}a\overline{c}$ and $ca\overline{a}d\overline{d}\overline{c}$ have to be in H, because they can be generated from $cL\overline{c}$ or $cM\overline{c}$. Looking at $ca\overline{a}d\overline{d}\overline{c}$ we can easily see that $(c, d\overline{d}\overline{c})$ is in $C_H(a\overline{a})$. According to the second condition of Theorem 1.1 $(c, d\overline{d}\overline{c})$ must also be in $C_H(ab\overline{b}\overline{a})$. In that case $cab\overline{b}ad\overline{d}\overline{c}$ must be a word in H, which obviously is not the case. Therefore we can state that XML languages are not closed under union. Because we already know that XML languages are closed under intersection we also know as direct consequence of DeMorgan's theorem that XML languages cannot be closed under difference either.

2   One-Unambiguous Regular Languages

2.1   Introduction and Basics

Something we have not taken care of in the last section is another requirement of the XML specification: The one-unambiguity of content-models. Beside our interest in one-unamiguity that emerges from this circumstance, one-unambiguity helps to efficiently parse a document which should merit our attention by its own.

We will begin by explaining what unambiguity and one-unambiguity means in the context of regular expressions. Consider a regular expression and the set of words that can be generated by this regular expression. If it is determinable for each letter in each word by which exact symbol of the regular expression it must have been generated after having read the whole word, the regular expression is unambiguous. If this can be done even when analyzing the word letter by letter not knowing the rest of the word in advance, the regular expressions is one-unambiguous.

To formalize this we need to formally distinguish between different occurrences of the same character symbol in a regular expression. This is done by subscripting each character symbol of the regular expression with an integer which has to be unique for each sort of symbols. This process is called *marking*. We denote the set of marked symbols as Π, the set of unmarked symbols as Σ, a regular expression over unmarked symbols as E, a regular expression over marked symbols as E', and the unmarking function as $\natural$.

Definition Let t, u, v, w be words over Π and $x, y \in \Pi$. A regular expression E is *one-unambiguous* iff $uxv, uyw \in L(E') \wedge x \neq y \Rightarrow x^\natural \neq y^\natural$. We say a language is one-unambiguous if it can be denoted by an one-unambiguous regular expression.

This definition is not well suited for the construction of a method to determine automatically if a given regular expression is one-unambiguous. We will therefore provide an alternative characterization of one-unambiguity for regular expressions on top of which we will construct a so called Glushkov automaton, which will help us in this matter.

Definition Let E be a regular expression. Then $first(E)$ is the set of all symbols that occur in any word of E at the first position, $last(E)$ is the set of all symbols that occur in any word of E at the last position, and $follow(E, a)$ is the set of all symbols that directly follow a in any word of E.

Theorem 2.1 *A regular expression E is one-unambiguous iff*

1. $\forall x, y \in first(E') : x \neq y \Rightarrow x^\natural \neq y^\natural$,

2. $\forall z \in sym(E') \wedge x, y \in follow(E', z) : x \neq y \Rightarrow x^\natural \neq y^\natural$,

where $sym(E')$ is the set of symbols occurring in E'. $\qquad\Box$

Consider *first*, *last* and *follow* as given in the last definition. The *Glushkov automaton* $G_E = (Q_E, \Sigma, \delta_E, q_I, F_E)$ of a regular expression E (where E' is a marking of E) is constructed as follows.

1. $Q_E :=$ all symbols of E' and a new, initial state q_I,

2. for $a \in \Sigma$: $\delta_E(q_I, a) := \{x \mid x \in first(E'), x^\natural = a\}$,

3. for $x \in sym(E')$ and $a \in \Sigma$: $\delta_E(x, a) = \{y \mid y \in follow(E', x), y^\natural = a\}$,

4. $F_E = \begin{cases} last(E') \cup \{q_I\}, & \text{if } \varepsilon \in L(E) \\ last(E'), & \text{otherwise.} \end{cases}$

Starting in q_I all states corresponding to a symbol that may occur as the first symbol of any word of E' are reachable by entering the corresponding (unmarked) symbol. For example, if $E = a^*b$ and $E' = a_1^*b_1$ then the state a_1 is reachable from q_I by entering a and b_1 by entering b. The same principle holds for *follow*. If the automaton is in a state corresponding to some symbol c_1, all states corresponding to a symbol that may directly follow c_1 in some word of the regular expression are reachable by entering the corresponding (unmarked) symbol. If the automaton reaches a state whose corresponding symbol is in the *last* set it will accept the word. If the empty word is part of the language q_I will also be accepted.

This way the Glushkov automaton of a regular expression E accepts all words that can be generated by E. Furthermore one-ambiguity manifests itself by having the opportunity to choose between different subsequent states for the same (unmarked) symbol. Consider $E = ba^*a$ and its marking $E' = b_1a_1^*a_2$ for example. If the automaton is in the state b_1 and an a is entered, it has to choose between the subsequent states a_1 and a_2 (and it is therefore undiscernible if an a that occurs in a word at this position has been generated by a_1 or a_2, hence the regular expression cannot be one-unambiguous). This leads to the following theorem.

Theorem 2.2 *A regular expression E is one-unambiguous iff G_E is a DFA.*□

In addition a Glushov automaton can be computed in time quadratic in the size of E. This means we have found a way to efficiently decide if a given regular expression is one-unambiguous. Whilst this is quite pleasing for now, we are far more interested in the recognition of one-unambigous regular languages than of mere regular expressions. This will be dealt with in the next subsection.

2.2 Recognition

After we were successful using automata to recognize one-unambiguous regular expressions in the last subsection, we will continue to do so on our quest for a method to recognize one-unambiguous regular languages. But first we need to introduce some definitions.

Definition For q being a state of an NFA, the *orbit* of q (denoted by $\mathcal{O}(q)$) is the strongly connected component of q.

This means that every state q' which can be reached from q and from which q can be reached is in the orbit of q.

Definition If $q \in F$ or if there is $q' \notin \mathcal{O}(q) : ((q,a),q') \in \delta$, then q is a *gate* of $\mathcal{O}(q)$.

This means that q is a gate of its orbit $\mathcal{O}(q)$ if it is a final state or if it has an outgoing transition to a state outside $\mathcal{O}(q)$.

Definition An NFA has the *orbit property* if all gates of each orbit have identical connections to the outside world.

"Identical" refers not only to the begin and end state of a transition but also to the label of a transition, of course.

If we restrict for a given automaton the set of states to $\mathcal{O}(q)$, set q as the initial state, and set the gates of $\mathcal{O}(q)$ as the final states, then the resulting automaton is called the *orbit automaton of q* and denoted by M_q. The language accepted by M_q is what we call the *orbit language of q*.

Now we can make our first shot at grasping the one-unambiguity of regular languages.

Theorem 2.3 *Let M be a minimal DFA. If and only if M has the orbit property and all orbit languages of M are one-unambiguous, then $L(M)$ is one-unambiguous.* □

In the case of the orbits being *trivial* (i. e. they consist only of one state and have no transitions) we can easily determine the one-unambiguity of the orbit languages. In all other cases however we must find a way to break down the orbits. For this we need two more definitions.

Definition For a DFA M, a symbol a in Σ is *M-consistent* if there is a state $f(a)$ in M such that all final states of M have an a-transition to $f(a)$. A set S of symbols is M-consistent if each symbol in S is M-consistent.

This means that a symbol a is M-consistent if M has a state which has incoming a-transitions from all final states.

Definition Let S be a set of symbols and M be an NFA. The *S-Cut of M* (denoted by M_S) is constructed from M by removing all a-transitions that leave a final state of M for each $a \in S$.

By choosing S as an M-consistent set we can extend Theorem 2.3 as follows.

Theorem 2.4 *Let M be a minimal DFA and S be an M-consistent set of symbols. If and only if M_S satisfies the orbit property and all orbit languages of M_S are one-unambiguous, then $L(M)$ is one-unambiguous.* □

This can be further extended to the following decision algorithm.

```
boolean one-unambiguous (MinimalDFA M) {
    compute S := {a ∈ Σ | a is M-consistent};
    if (M has a single, trivial orbit) {return true;}
    if (M has a single, nontrivial orbit && S = ∅) {return false;}
    compute the orbits of M_S;
    if (!OrbitProperty(M_S)) {return false;}
    for (each orbit K of M_S) {
        choose x ∈ K;
        if (!one-unambiguous((M_S)_x) {return false;}
    }
    return true;
}
```

2.3   Closure

As in the last section it is not within the scope of this assignment to discuss closure properties in detail. So again we will give the results right away.

Definition Let L be a language and w be a word. The *derivative* of L with respect to w (denoted by $w \backslash L$) is defined as $\{v \mid wv \in L\}$.

- The family of one-unambiguous regular languages is closed under derivatives.

- One-unambiguous regular expressions are not closed under derivatives, unless they are in a star normal form.

- The family of one-unambigous regular languages is not closed under union, concatenation or star.

3   Analysis of XML Schema Languages

3.1   Introduction and Basics

As has been said in the introduction to the first section, XML schema languages are often needed to impose constraints beyond the constraints XML provides itself. Whilst DTDs (which we examined in the first section) are native to the XML specification, there have been other proposals of XML schema languages like XML Schema, RELAX, RELAX NG, DSD, and XDuce. Somebody trying to specify a document type is naturally interested in which XML schema languages are suited to describe the constraints he has in mind. Furthermore when specifying the successor to an already existing document type the question may arise, which documents of the old type are also valid under the new document specification. And last but not least we want to be able to efficiently check if a document conforms to a document specification. This means we are interested in a comparison of XML schema languages regarding expression power, closure properties and document validation qualities. To achieve such a comparison,

we will have a look at the language classes corresponding to the different XML schema language proposals. As usual we have to start with introducing some terms and definitions.

A *model group* is a regular expression in which the additional operators ?, &, and $^+$ are allowed. $E?$ denotes $L(E + \varepsilon)$, $F\&G$ denotes $L(FG + GF)$, and E^+ denotes $L(EE^*)$.

A *regular tree grammar* is a grammar, whose production rules have the form $X \rightarrow a\ Expression$ where X is a non-terminal symbol, a is terminal symbol denoting the root of the (sub)tree and *Expression* is a model group over non-terminal symbols denoting the children of the root. Please note that regular tree grammars (as opposed to XML grammars) can have multiple start symbols.

If we distinguish for a regular tree grammar which non-terminal symbols are used for deriving trees and which non-terminal symbols are used for content-model specification and adjust the set of production rules accordingly, this will lead us to a grammar in *normal form 1 (NF1)*.

Definition A grammar $G = (N1, N2, T, P1, P2, S)$ with

- T being the set of terminal symbols and S being the set of start symbols,

- $N1 =$ non-terminal symbols used for deriving trees,

- $N2 =$ non-terminal symbols used for content-model[1] specification,

- $P1 =$ productions of the form $A \rightarrow aX$ with $A \in N1, X \in N2$,
  $a \in T$ (only one production per symbol $\in N1$),

- $P2 =$ productions of the form $X \rightarrow Exp$ with $X \in N2$, *Exp* model
  group over $N1$ (only one production per symbol $\in N2$).

is in *normal form 1 (NF1)*.

3.2   Language Classes

Let us now introduce two language classes important for the evaluation of XML schema languages.

Definition If in a grammar there is no more than one rule of the form $A \rightarrow aX$ in $P1$ for all non-terminal symbols a, we call this grammar a *local tree grammar* and the corresponding languages *local tree languages*.

Definition Two different non-terminals A and B are called competing with each other if

- one production rule has A on the left-hand side,

- another production rule has B on the left-hand side, and

---

[1]$contentModel(A)$ $(A \in N1)$ is the model group over $N1$ denoting the content of A.

  • these two production rules share the same terminal on the right-hand side.

Definition We call a grammar *single-type constraint grammar* if for each production rule, non-terminals in its content model do not compete with each other and if start symbols do not compete with each other. Languages generatable by such grammars are called *single-type constraint languages.*

Single-type constraint languages and local tree languages are closed under intersection, not closed under union, and not closed under difference.

Theorem 3.1 *Local tree languages form a proper subclass of single-type constraint languages.* $\square$

Proof A local tree language is a single-type constraint language by definition. Now consider a regular tree grammar with $A, B \in N1 \wedge A \neq B \wedge root(A) = root(B)$. This grammar can be a single-type constraint grammar but it cannot be a local tree grammar. Hence there are single-type constraint languages which are not local tree languages.

3.3   Evaluating XML Schema Languages

We are now going to determine how the XML schema languages DTD, DSD, XML Schema, and RELAX relate to (foremost these two) language classes. Do not be irritated by the mention of TDLL(1) and TD(1) grammars which we have not covered, because—as we will see very soon—they do not matter for the comparison we try to accomplish.

  • DTDs are TDLL(1) grammars which are also local tree grammars.

  • DSDs do not impose constraints on the production rules. This means that in theory any regular tree grammar can be expressed in DSD. Practically however the parsing algorithm uses a greedy technique with one vertical and horizontal lookahead which leads to the assumption that all and only TDLL(1) languages are accepted.

  • XML Schema corresponds to TDLL(1) grammars which are also single-type constraint grammars. Furthermore, group definitions are allowed to contain other group definitions without restriction which would make context-free content models possible. Because this is very unlikely to be intentional, a specification mistake is suspected here.

  • RELAX corresponds to arbitrary regular tree grammars.

Regarding the comparison of XML Schema and DTD we have shown in Theorem 3.1 that more can be expressed in XML Schema (because XML Schema and DTD both are TDLL(1), TDLL(1) does not matter for the comparison). It is trivial to show that there are regular tree grammars which are not single-type or local tree grammars. Therefore RELAX has to have more expression power

than either of XML Schema or DTD.[2] Regarding DSD we might add (without proof) that there are TDLL(1) languages which are not single-type constraint languages and therefore—if the assumption that DSD practically corresponds to TDLL(1) languages is true—DSD has more expression power than DTD and XML Schema, but less than RELAX (and XDUCE). These results become more clear in the following figure, which has been taken from [3].
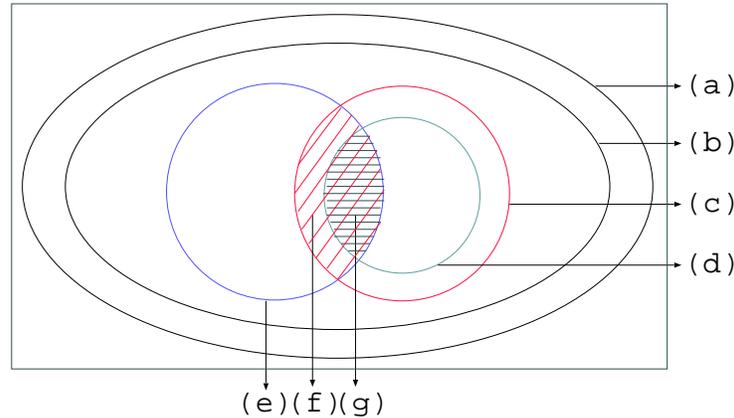
(a) = regular tree grammars (RELAX, XDuce), **(b)** = TD(1) grammars, **(c)** = single-type constraint grammars, **(d)** = local tree grammars, **(e)** = TDLL(1) grammars (DSD?), **(f)** = TDLL(1) and single-type constraint grammar (XML Schema), **(g)** = TDLL(1) w/ and local tree grammars (DTD)

References

[1] Jean Berstel, Luc Boasson. *Formal Properties of XML Grammars and Languages.* Acta Informatica, 38:649–671, 2002.

[2] Anne Brüggemann-Klein, Derick Wood. *One-Unambiguous Regular Languages.* Information and Computation, 140:229–253, 1998.

[3] Dongwon Lee, Murali Mani, Makoto Murata. *Reasoning about XML Schema Languages using Formal Language Theory.* Technical Report, IBM Almaden Research Center, 2000. Log #95071.

[4] Thomas Schwentick. *Formal Methods for XML: Algorithms & Complexity.* Internet: <http://lrb.cs.uni-dortmund.de/~tick/Talks/edbtp.pdf>, 2004 (cited 2006–01–26).

[5] Anders Møller, Michael I. Schwartzbach. *The XML Revolution: Technologies for the future Web.* Internet: <http://www.brics.dk/~amoeller/XML/>, 2003 (cited 2006–01–26).

---

[2]This is also true for XDUCE, which we have not covered above.

[6] Dongwon Lee, Murali Mani, Makoto Murata. *Taxonomy of XML Schema Languages Using Formal Language Theory*. Proceedings of the 2001 Conference on Extreme Markup Languages, 2001.