On validation of XML streams using finite state machines

Cristiana Chitic
Department of Comp. Science
University of Toronto
Toronto, ON, Canada
cristi@cs.toronto.edu

Daniela Rosu
Department of Comp. Science
University of Toronto
Toronto, ON, Canada
drosu@cs.toronto.edu

ABSTRACT

We study validation of streamed XML documents by means of finite state machines. Previous work has shown that validation is in principle possible by finite state automata, but the construction was prohibitively expensive, giving an exponential-size nondeterministic automaton. Instead, we want to find deterministic automata for validating streamed documents: for them, the complexity of validation is constant per tag. We show that for a reading window of size one and nonrecursive DTDs with one-unambiguous content (i.e. conforming to the current XML standard) there is an algorithm producing a deterministic automaton that validates documents with respect to that DTD. The size of the automaton is at most exponential and we give matching lower bounds. To capture the possible advantages offered by reading windows of size k, we introduce k-unambiguity as a generalization of one-unambiguity, and study the validation against DTDs with k-unambiguous content. We also consider recursive DTDs and give conditions under which they can be validated against by using one-counter automata.

1. INTRODUCTION

As an increasing number of organizations and individuals are using the Internet, the ability to manipulate information from various sources is becoming a fundamental requirement for modern information systems. The XML data format has been adopted as the common format for data exchange on the Web, and, to facilitate query answering, XML data needs to be validated against DTDs and XML-Schemas.

Streamed data is data originating from sources that provide a continuous flow of information. Storing the flowing data stream in order to process it is not a feasible solution since it would normally require large amounts of memory. To cope with this issue, one needs to find ways to process data as it comes without going back and forth in the stream. The symbols in the stream are the element tags and the included text in the order of their appearance in the XML document. Since we are interested in checking only structural properties, such as DTD conformance, we ignore the data values and consider the stream as a sequence of opening and closing tags. Checking that an XML stream conforms to a DTD in a single pass under memory constraints is referred to as the *on-line validation* of streamed XML documents [13]. There are two types of on-line validation considered in [13]: *strong validation*, which includes checking well-formedness of the input document, and *validation*, which assumes that the input is a well-formed document.

In this work, we investigate on-line validation of XML documents using finite-state machines. The nonrecursive DTDs conforming to the current standard [14] (i.e. with one-unambiguous content) correspond to a reading window of size one. Further, we introduce k-unambiguous regular expressions as a generalization of one-unambiguous regular expressions and study nonrecursive DTDs with k-unambiguous content. In both cases the finite machines are deterministic. The advantage is having constant time complexity of validation for each open/closed tag. We prove exponential lower and upper bounds for the size of the minimal deterministic automaton used for strong validation.

According to statistical data [6], nonrecursive DTDs are more frequent than recursive DTDs, but still recursive DTDs are used commonly in practice. For on-line validation against recursive DTDs we propose using one-counter automata. We also give syntactic conditions under which recursive DTDs can be recognized by one-counter automata.

Related Work On-line validation of streamed XML documents under memory constraints has also been studied in [13]. One of the results there is that for any nonrecursive DTD one can construct a finite automaton (a particular case of a result in [12]). An algorithm was given that constructs for any nonrecursive DTD a finite automaton that can be used to perform strong validation. However, the resulting automaton was nondeterministic, exponential in the size of the DTD and had exponential (in the size of the DTD) per-tag complexity of validation. Validating against recursive DTDs was also considered in [13]. Under the restrictive assumption that the input stream is well-formed, they present a class of recursive DTDs validated by finite automata.

Strong validation against recursive DTDs can be performed by push-down automata. However, their disadvantage is having a stack of size proportional, in the worst case, to the size of the XML stream. This approach was discussed in [13, 10]. In our approach the space required by the counter is logarithmic in the size of the input stream.

One-unambiguous regular expressions [4] reflect the requirement that a symbol in the input word be matched uniquely to a position in the regular expressions without looking ahead in the word. We generalize this concept in a different way than in the extension considered in [7]. With a lookahead of k symbols we want to determine the next, unique, matching position in the regular expression, while in the approach considered in [7] a lookahead of at most k symbols will determine uniquely the next k positions.

We assume the reader is familiar with basic notions of language theory: (nondeterministic) deterministic finite-state automaton ((N)DFSA), context-free grammar (CFG) and language (CFL), extended context-free grammar (ECFG)(e.g., see [2, 12]).

The paper is organized as follows. The first section describes the problem of validating XML streams against DTDs. Section 2 presents canonical XML-grammars associated to any DTD and also the size of the minimal automaton used for validating nonrecursive DTDs. In Section 3 we establish results for DTDs with one-unambiguous and k-unambiguous content. Finally, in Section 4 we investigate strong validation against recursive DTDs.

2. THE VALIDATION PROBLEM

Let Σ be a finite alphabet. An XML document is abstracted as a tree document. A *tree document* over Σ is a finite ordered tree with labels in Σ. Formally, a *string representation* denoted $[t]$ is associated to each tree document t as follows: if t is a single node labeled a, then $[t] = a\,\bar{a}$; if t consists of a root labeled a and subtrees $t_1 ... t_k$ then $[t] = a[t_1]...[t_k]\bar{a}$, where a and $\bar{a}$ are opening and closing tags. Let $\overline{\Sigma} = \{\bar{a} | a \in \Sigma\}$ denote the alphabet of closing tags. An XML document is a *well-formed* document if the string representation corresponding to the XML tree is well-balanced. If T is a set of tree documents, $\mathcal{L}(T)$ denotes the language consisting of the string representations of the tree documents in T.

A *DTD* (*Document Type Definition*) [14] $D = (\Sigma, R, r)$ that ignores the attribute elements is a finite set of rules R of the form $a \rightarrow R_a$ such that $a \in \Sigma$, R_a is a regular expression over Σ and $r \notin \Sigma$ is called the *root*. The set of tree documents satisfying a DTD D is denoted by $SAT(D)$. The language over $\Sigma \cup \overline{\Sigma}$ consisting of the string representations of all tree documents in $SAT(D)$ is defined as: $\mathcal{L}(D) = \{[t] \mid t \in SAT(D)\}$. The *dependency graph* D_g of a DTD D is the graph whose set of nodes is Σ, and for each rule $a \rightarrow R_a$ in the DTD there is an edge from a to b for each b occurring in some word in R_a. A DTD is *nonrecursive* if and only if D_g is acyclic and is *recursive* if and only if D_g is cyclic.

The problem of validating an XML stream with respect to a DTD D is defined as checking that the string representation of the XML document is contained in the associated language $\mathcal{L}(D)$.

3. CANONICAL XML-GRAMMARS

In the context of streaming, since the attributes are ignored, a DTD appears to be a special kind of extended context-free grammar. The formal grammar that captures explicitly the opening and closing tags is the *XML-grammar*, first introduced in [1]. Given a DTD $D = (\Sigma, R)$ we denote a corresponding XML-grammar by $D_{ECF} = (N, \Sigma \cup \overline{\Sigma}, ROOT, P)$, where $\Sigma \cup \overline{\Sigma}$ is the set of terminals, N is the set of nonterminals and N is in 1-1 correspondence with Σ, $ROOT$ is the start symbol and P is the set of productions. The set P contains only rules of the following types: $ROOT \rightarrow r\, R_{ROOT}\, \bar{r}$; $A \rightarrow a\, R_A\, \bar{a}$; $A \rightarrow a\, \bar{a}$, where $A \in N$, $a \in \Sigma$ and $\bar{a} \in \overline{\Sigma}$. R_{ROOT}, R_A is a regular expression containing only nonterminals and corresponds to the nonterminal $ROOT$, A respectively.

Since XML-grammars [1] were studied without providing a way to link them with DTDs, we give an algorithm that transforms a DTD D into an XML-grammar D_{ECF} such that $L(D_{ECF})$, i.e. the language of the grammar D_{ECF}, is the same as $\mathcal{L}(D)$. We call the grammar produced by the algorithm the *canonical XML-grammar* associated to the DTD D. Let $D = (\Sigma \cup \overline{\Sigma}, r, R)$ be a DTD, Π be an alphabet such that $\Pi \cap \Sigma = \emptyset$ and $ROOT$ be the symbol such that $ROOT \notin \Pi$. Π is the alphabet of nonterminals in the canonical XML-grammar. Let $f : RegExp(\Sigma) \cup \{r\} \rightarrow RegExp(\Pi) \cup \{ROOT\}$ be a function such that $f(r) = ROOT$, $f(\Sigma) = \Pi$ and $f_{/E}$ is a bijection. The set of terminals of the canonical XML-grammar is $T = \Sigma \cup \overline{\Sigma}$. The set of productions P of the canonical XML-grammar are modifications of the rules of the DTD where rules of the form $a \rightarrow R_a$ are transformed into productions of the form $f(a) \rightarrow af(R_a)\bar{a}$. The output is the canonical XML-grammar $D_{ECF} = (N, T, ROOT, P)$.

The *canonical XML-grammar* associated to a DTD is instrumental in proving the results of this paper.

Example 3.1. Consider the DTD D over $\Sigma = \{r, a, b, c\}$ with the rules: $r \rightarrow a^*b$, $a \rightarrow bc$, $b \rightarrow c + \epsilon$, $c \rightarrow \epsilon$. The algorithm gives the XML-grammar $D_{ECF} = (N, \Sigma \cup \overline{\Sigma}, ROOT, P)$, where $N = \{ROOT, A, B, C\}$ and the set of productions P is: $\{ROOT \rightarrow rA^*B\bar{r}$, $A \rightarrow aBC\bar{a}$, $B \rightarrow b(C + \epsilon)\bar{b}$, $C \rightarrow c\bar{c}\}$.

Given a DTD D, the language $\mathcal{L}(D)$ is the same as the language generated by the canonical XML-grammar that corresponds to D. Thus, validating an XML stream with respect to a DTD D becomes equivalent to checking that the stream belongs to $L(D_{ECF})$, which is the definition we will use throughout the paper.

The canonical XML-grammar D_{ECF} corresponding to a nonrecursive DTD D is nonrecursive, thus the language $L(D_{ECF})$ is regular [12]. We give an algorithm of *bottom-up substitution* that takes as input a canonical nonrecursive XML-grammar D_{ECF} and returns a regular expression that generates the language $L(D_{ECF})$ [5].

The regular expression that generates $L(D_{ECF})$ is the regular expression corresponding to the nonterminal $ROOT$, computed by the *bottom-up substitution* algorithm.

Example 3.2. Let D and D_{ECF} be the DTD, respectively the XML-grammar grammar from the previous example. The regular expression obtained by bottom-up substitution is $r(a(b(c\bar{c} + \epsilon)\bar{b}c\bar{c})\bar{a})^*(b(c\bar{c} + \epsilon)\bar{b})\bar{r}$.

The regular expression obtained by applying *bottom-up substitution* to a D_{ECF} is used to construct an automaton which is used as a validating tool. The size of the automaton is linear in the size of the expression.

An algorithm was presented in [13], which yields a validating non-deterministic automaton whose size is exponential in the size of the DTD. Here we show that even for the minimal automata the lower bound on their size is still exponential with respect to the size of the DTD. In order to compute the lower bound we partition the symbols of the alphabet into strata, defined below.

Definition 3.3. Stratum 0 *of a nonrecursive DTD is the set of all symbols in Σ such that the right-hand sides of the corresponding rules contain only the symbol ϵ.* Stratum i *of a DTD is the set of all symbols in Σ such that the right-hand sides of the corresponding rules contain only symbols from Stratum 0, ..., Stratum $i-1$ with at least one symbol from Stratum $i-1$.*

We define the *depth* of a nonrecursive DTD to be the corresponding number of strata. We parameterize the *size* of a nonrecursive DTD by its *depth* and the *maximum length* of a right-hand side of a rule.

Remark 3.4. For a nonrecursive DTD, the number of strata is finite and a symbol in the alphabet belongs to only one stratum. Also, in general, the *depth* of a DTD is not related to the number of productions.

Let D be a nonrecursive DTD. Let d be its depth and let L be the maximum number of symbols that appear in the body of a rule.

Lemma 3.5. *[5] The size of the minimal automaton A_D that recognizes $\mathcal{L}(D)$ is bounded from above by $1 + 2 \cdot \frac{L^d-1}{L-1}$.*

We show that the bound is tight by considering the following example.

Example 3.6. Let $D = (\{r, a_1, ..., a_n\}, R)$ be a nonrecursive DTD s.t. the set of rules is $R = \{r \to a_1\, a_1, a_1 \to a_2\, a_2, ..., a_{n-1} \to a_n\, a_n, a_n \to \epsilon\}$. The regular expression corresponding to this DTD obtained by applying the *bottom-up substitution* algorithm to the associated D_{ECF} is $r\,(a_1\,(a_2\,(...\,(a_{n-1}\,(a_n\,\overline{a}_n)^2\,\overline{a}_{n-1})^2\,...)^2\,\overline{a}_2)^2\,\overline{a}_1)^2\,\overline{r}$. The minimal automaton that recognizes $\mathcal{L}(D)$ has $1 + 2 \cdot (2^{n+1} - 1)$ states, where $n+1$ is the depth of D and the maximum length of a production in R is 2.

4. STREAMED VALIDATION OF NONRECURSIVE DTDS

We now investigate the problem of validating XML streams against nonrecursive DTDs. The XML standard [14] requires that the regular expressions associated with each production match uniquely a position of a symbol in the expression to a symbol in an input word without looking beyond that symbol. This scenario corresponds to processing the stream with a deterministic automaton having a reading window of size one. The regular expressions described by the standard are the one-unambiguous regular expressions introduced in [4].

Before presenting the results of this section, we recall the definition and some characterizations of one-unambiguous regular expressions [4]. To denote different occurrences of the same symbol in an expression, all the symbols are marked with unique subscripts. For example, a *marking* of the expression $b((a^+bc)^*|(abd)^*)$ is $b_7((a_1{}^+b_2c_3)^*|(a_4b_5d_6)^*)$. The set of symbols of expression E is denoted by $sym(\Sigma)$. For expression E, we denote its marking by E'. Each subscripted symbol is called a *position.* For a given position x, $\mathcal{X}(x)$ indicates the corresponding symbol in Σ without the subscript. Formally, an expression E is defined to be *one-unambiguous* if and only if, for all words u, v, w over $sym(E')$ and all symbols x,y in $sym(E')$, if the words uxv, $uyw \in L(E')$ and $x \neq y$ then $\mathcal{X}(x) \neq \mathcal{X}(y)$. One possible method to convert a regular expression into a finite automaton was proposed in [8]. In the *Glushkov automaton* of an expression E, the states correspond to positions in E and the transitions connect positions that are consecutive in a word in $L(E')$. For each expression E, the following sets are defined: $First(E')$, the set of positions that match the first symbol of some word in $L(E')$; $Last(E')$, similarly for the last positions; and $Follow(E', z)$, the set of positions that can follow position z in some word in $L(E')$. A *Glushkov automaton* corresponding to a regular expression is constructed using the sets defined above. The Glushkov automaton has as many states as the number of positions in the corresponding marked expression plus one.

Definition 4.1. *A DTD D (an XML-grammar D_{ECF}) is* one-unambiguous *if all the rules (productions) have one-unambiguous regular expression in their right-hand sides.*

The canonical XML-grammar associated to a one-unambiguous DTD is also one-unambiguous.

Theorem 4.2. *Let $D = (\Sigma, R)$ be a nonrecursive one-unambiguous DTD. Then the language $\mathcal{L}(D)$ is one-unambiguous.*

To prove that the language $\mathcal{L}(D)$ (or the language $L(D_{ECF})$ generated by the canonical grammar D_{ECF}) is one-unambiguous it is sufficient to show that there exists a one-unambiguous expression that generates the language. The solution to this problem is the regular expression obtained by bottom-up substitution [5].

Corollary 4.3. *There is an algorithm that, given a nonrecursive one-unambiguous DTD D, constructs a deterministic Glushkov automaton G_D such that G_D accepts precisely the language consisting of the string representations of the documents that conform to D.*

The algorithm in [13] constructs an exponential-size *nondeterministic* automaton, which yields per open/closed tag complexity that is exponential in the size of the DTD. In contrast, for one-ambiguous DTDs, we can construct a *deterministic* automaton that verifies conformance to the DTD for streaming documents. This yields constant per open/closed tag complexity.

Remark 4.4. *If the validation of nonrecursive DTDs is performed using their corresponding Glushkov automata, the exponential lower and upper bounds of these automata in the size of the DTDs are the same as the bounds shown in the previous section.*

From example 3.6 and remark 4.4 is follows that the exponential bound on the size of the Glushkov automata accepting $\mathcal{L}(D)$ for a *one-unambiguous* DTD D cannot be improved. The lower bound on the size of the minimal deterministic automata accepting $\mathcal{L}(D)$ is also exponential in the size of D.

We consider now the case of DTDs with non-deterministic content, which appear in practice in a variety of fields [6, 5]. To model these types of DTDs we introduce the *k-unambiguous* regular expressions. They are a generalization of *one-unambiguous* regular expressions, different from the one considered in [7]. Informally, a k-unambiguous expression matches uniquely a position of a symbol in the expression to a symbol in an input word by looking ahead k symbols. Practically, an XML stream can be validated against a nonrecursive DTD with k-unambiguous content by a finite automaton that uses a reading window of size k to move to a unique state.

Definition 4.5. *A regular expression E is* k-unambiguous, *where $k \leq |E|$, if and only if for all words u, v, w, $x = x_1 \dots x_k$ and $y = y_1 \dots y_k$ over $sym(E')$, $ux_1 \dots x_k v \in L(E')$, $uy_1 \dots y_k w \in L(E')$ and $x_k \neq y_k$ implies that $\mathcal{X}(x_1) \dots \mathcal{X}(x_k) \neq \mathcal{X}(y_1) \dots \mathcal{X}(y_k)$.*

Example 4.6. Consider the regular expression $b((a^+bc)^*|(abd)^*)$ marked as $b_7((a_1{}^+b_2c_3)^*|(a_4b_5d_6)^*)$. This regular expression is not one-unambiguous. Given the word *babc* it cannot be decided if the symbol a following the symbol b is matching position a_1 or a_4. By looking ahead *4* symbols we can match the occurrence of symbol c with position c_3.

Example 4.7. The following 3-unambiguous content model $stand\_point \rightarrow ((back\_sight, fore\_sight|back\_sight, fore\_sight, back\_sight), intermidiate\_sight^*, info\_stand?, info\_i^*)$ appears in a DTD that describes digital measurements *http:// gama.fsv.cvut.cz/˜soucek/ dis/ convert/ dtd/ dnp-1.00.dtd.*

Another way of characterizing *k-unambiguous* regular expressions is using Glushkov automata.

Definition 4.8. *A Glushkov automaton $G = (Q, \Sigma, \delta, F)$ is k-deterministic if for every state $p \in Q$ and every word $a_1 \dots a_k$ over Σ, the extended transition $\delta^*(p, a_1 \dots a_k)$ contains at most one state.*

In other words, we call a Glushkov automaton *k-deterministic* if from any state following all paths labeled $a_1 \dots a_k$ we reach at most one state.

Given a marked expression E' we define the sets: $First(E', k) = \{w \in sym(E')|$ there is a word u s.t. $wu \in L(E')$ and $|w| = k\}$, $Follow(E', z, k) = \{w \in sym(E')|$ there are words u and v s.t. $uzwv \in L(E')$ and $|w| = k\}$, for all symbols $z \in sym(E')$ and $Last(E', k) = \{ w \in sym(E')|$ there is a word u s.t. $uw \in L(E')$ and $|w| = k\}$. These sets can be computed in time polynomial in the size of E', and thus we can give a polynomial-time algorithm to check whether a regular expression is *k-unambiguous*, for a given k. The worst case time complexity is $O(|E'|^{k+1})$. There are regular expressions that are not k-unambiguous for any number k.

Example 4.9. The Glushkov automaton in *Figure 1(i)* is k-deterministic for any $k \in \mathbb{N}$, since $\delta^*(a_1, aaaaaaaaa) = \{a_1, a_3\}$. The Glushkov automaton in *Figure 1(ii)* is $3-$ *unambiguous*.

Proposition 4.10. *(a) If a regular expression E is k-unambiguous then E is $k+1$-unambiguous. (b) A regular expression E is k-unambiguous if and only if the Glushkov automaton corresponding to E is k-deterministic.*

Given a nonrecursive DTD $D = (\Sigma, R)$, we obtain by applying our *bottom-up substitution* algorithm to the canonical XML-grammar D_{ECF} a regular expression E_D that describes the language $L(D_{ECF}) = \mathcal{L}(D)$. The Glushkov automaton corresponding to E_D is used for validation XML streams against D. If the regular expression E_D is *k-unambiguous*, then the automaton used for online validation is also k-deterministic. We define the set

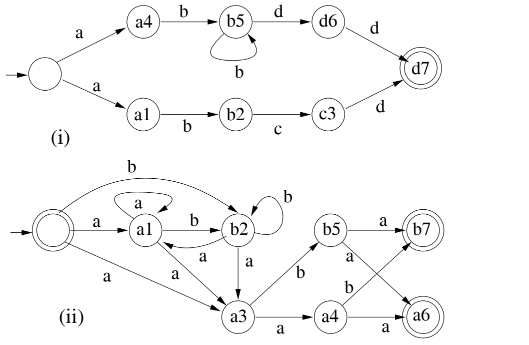

Figure 1: Glushkov automaton corresponding to the regular expressions (i): $(a+b)^*a(a+b)(a+b)$ and (ii): $(abc+ab^*d)d$.

$\Sigma_k = \{ a \in \Sigma| \, a \rightarrow R_a \in R$ and R_a is k -unambiguous$\}$ for any number k. A regular expression is called *finite* if it denotes a finite language. Now, we define the set $\Sigma_k^{fin} = \{ a \in \Sigma \mid a \rightarrow R_a \in R$ and the regular expression R_a is finite k-unambiguous$\}$ for any number k. On the dependency graph D_g we define the set $Reachable_a$, $a \in \Sigma$, to be the set of symbols contained in the subtree rooted in a.

As shown in the example 4.9, there exist expressions that are not *k-unambiguous* for any number k. Also, similarly to the class of *one-unambiguous* expressions, the class of *k-unambiguous* expressions is not closed under union, concatenation and star operation. Thus, we need to impose conditions on the nonrecursive DTDs that guarantee that the *bottom-up substitution* algorithm applied to the corresponding XML-grammars yields a *k-unambiguous* expression.

Theorem 4.11. *Let $D = (\Sigma, R)$ be a nonrecursive DTD with the root rule denoted by $r \rightarrow R_r$. Let D_g be the dependency graph of D. Assume that one of the following conditions is true:*

1. The regular expression R_r is k-unambiguous and $Reachable_a \subseteq \Sigma_p^{fin}$ for $a \in sym(R_r)$

2. The regular expression R_r is one-unambiguous and there exists at most one $a \in \Sigma_k$ such that $Reachable_a \subseteq \Sigma_1^{fin}$, none of the elements on the path from a to the root appear under a Kleene-star and

$$a \in \bigcup_{b \in sym(R_r)} Reachable_b.$$

Then there exists a number k' such that the regular expression associated to the canonical XML-grammar D_{ECF} and obtained by bottom-up substitution is k'-unambiguous.

We illustrate the conditions of the theorem in the following two examples.

Example 4.12. Let $D = (\{r, a, b, c, d, e, f\}, \{r \rightarrow (a^+e|ac)(cd)^*, a \rightarrow b, b \rightarrow ce|cf, c \rightarrow \epsilon, d \rightarrow ccf|cce, e \rightarrow \epsilon, f \rightarrow \epsilon\})$ be a nonrecursive DTD satisfying the first condition of the theorem. The expression R_r is 2-unambiguous and the rest of the rules correspond to finite 3-unambiguous expressions. The regular expression obtained by the algorithm of bottom-up substitution is $r((ab(c\bar{c}e\bar{e}|c\bar{c}f\bar{f})\bar{b}\bar{a})^+e\bar{e}|ab(c\bar{c}e\bar{e}|c\bar{c}f\bar{f})\bar{b}\bar{a}c\bar{c})(c\bar{c}d(c\bar{c}c\bar{c}f\bar{f}|c\bar{c}c\bar{c}$ $e\bar{e})\bar{d})^*\bar{r}$, where the maximum of the lengths of the regular expressions associated to the nonterminals A, B, C, D, E, F is 14. Thus, $k' = 14$.

Example 4.13. Let $D = (\{r, a, b, c, d, e, f\}, \{r \to ba^+, a \to ed^*f, b \to c|d^+, c \to f^+e|fd, f \to ee, d \to \epsilon, e \to \epsilon\})$ be a nonrecursive DTD that satisfies the second condition of the theorem. The regular expression obtained by the algorithm of bottom-up substitution applied on the canonical XML-grammar associated to D is $rb(c((fe\bar{e}e\bar{e}\bar{f})^+e\bar{e}|fe\bar{e}e\bar{e}\bar{f}d\bar{d})\bar{c}|(d\bar{d})^+)\bar{b}(ae\bar{e}(d\bar{d})^*fe\bar{e}e\bar{e}\bar{f}\bar{a})^+\bar{r}$. In this case $k' = 18$, which is the length of the regular expression associated to the nonterminal C.

As the following examples show, the theorem is quite tight, since there are DTDs that deviate only slightly from the conditions of the theorem and have associated regular expressions that are not k-unambiguous.

Example 4.14. Let $D = (\{r, a, b, c, d\}, \{r \to ab|ac, a \to d^*, b \to \epsilon, c \to \epsilon, d \to \epsilon\})$ be a nonrecursive DTD, for which the first condition of the theorem is not satisfied. The regular expression associated to the DTD is $r(a(d\bar{d})^*\bar{a}b\bar{b}|a(d\bar{d})^*\bar{a}c\bar{c})\bar{r}$, which is not a k'-unambiguous regular expression for any $k' \leq 14$ (the length of the expression being 14).

Example 4.15. Let $D = (\{r, a, b, c, d, e\}, \{r \to ab, a \to cd|ce, b \to \epsilon, c \to d^*, d \to \epsilon, e \to \epsilon\})$ be a nonrecursive DTD, for which the second condition of the theorem is not satisfied. The regular expression associated to the DTD is $ra(c(d\bar{d}^*\bar{c}d\bar{d}|c(d\bar{d})^*\bar{c}e\bar{e})\bar{a}b\bar{b}\bar{r}$. The Glushkov automaton of this expression is not k'-unambiguous for any k', since the extended transition of the automaton contains more than one state when reading words $racdd\bar{d}d\bar{d}...d\bar{d}$ of arbitrary length.

5. STREAMED VALIDATION OF RECURSIVE DTDS

We now consider the problem of strongly validating an XML stream against a *recursive* DTD using (*restricted*) *one-counter automata*. Recursive DTDs appear in fields like: computational linguistics, web distributed data exchange, financial reposting, etc [5].

Example 5.1. The following recursive content model $exception \to (type, msg, contextdata^*, exception?)$ appears in a DTD from Workflow Management Coalition *http://www.oasis-open.org/cover/WFXML10a-Alpha.htm.*

A *one-counter automaton* (1-$CFSA$) is a quintuple $M = (Q, \Sigma, H, q_0, F)$, where Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, $F \subset Q$ is the set of *final states*, Σ is the finite *alphabet* and the *transition set* H is a finite subset of $Q \times (\Sigma \cup \epsilon) \times \mathbb{Z}^+ \times \{-1, 0, 1\} \times Q$ [9, 11]. Thus, the 1-$CFSA$ consists of a finite state automaton and a counter that can hold any nonnegative integer and can only test if the counter is zero or not. The move of the machine depends on the current state, input symbol and whether the counter is zero or not. In one move, M can change state, add +1, 0 or −1 to the counter. However, the counter is not allowed to subtract 1 from 0. The machine accepts a word if it starts in the initial state with the counter 0 and reaches a final state with the input completely scanned and the counter 0. A language accepted by a one-counter machine is called *one-counter language*. We denote the family of one-counter languages by OCL. A *restricted* one-counter automaton (*restricted*-1-$CFSA$) is a one-counter automaton which, during the computation cannot test if the counter is zero [9, 2, 3]. The language accepted by a restricted one-counter automaton is called *restricted one-counter language* and the family of such languages is denoted $ROCL$. It is known that the family of restricted one-counter languages are strictly included in the family of one-counter languages [3] and that OCL is $NSPACE(log\ n)$ [15]. Every language accepted by a one-counter automaton is a context-free language [2].

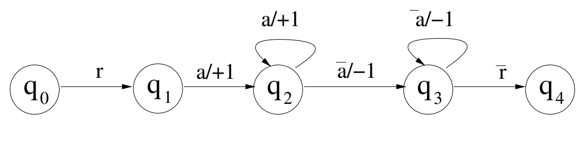

Figure 2: One-counter automaton recognizing the language $\{ra^n\bar{a}^n\bar{r}|n > 0\}$

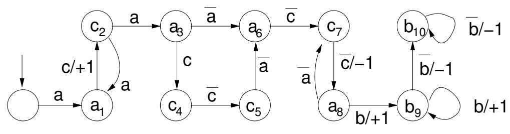

Figure 3: One-counter automaton recognizing the language $\{(ac)^n a(\epsilon|c\bar{c})\bar{a}(\bar{c}\bar{a})^n b^m\bar{b}^m\}$

Example 5.2. The language corresponding to the DTD $D = \{r \to a, a \to a|\epsilon\}$ is $ra^n\bar{a}^n\bar{r}$ and a streamed XML document can be strongly validated against D using the one-counter automaton in Figure 2.

Definition 5.3. *A DTD D is (deterministic) one-counter recognizable, (deterministic) restricted one-counter recognizable, if the language $\mathcal{L}(D)$ is in (D−)OCL, (D−)ROCL respectively. An XML-grammar D_{ECF} is (deterministic) one-counter recognizable, (deterministic) restricted one-counter recognizable, if the language $L(D_{ECF})$ is in (D−)OCL,(D−)ROCL respectively.*

We denote the family of *one-counter recognizable* (*restricted one-counter recognizable*) DTDs by $OCRD$ ($ROCRD$). The family of languages generated by (restricted) one-counter recognizable DTDs is contained strictly in the family $(R)OCL$, since there are languages that are *(restricted) one-counter* but cannot be generated by a DTD, e.g. $\{\alpha\sharp\beta|\ \alpha, \beta \in \Sigma^*, \sharp \notin \Sigma, |\alpha| \neq |\beta|\}$ [2].

The family of *strongly recognizable* DTDs [13] is included in $(R)OCRD$, since the regular languages are included in the (restricted) one-counter languages. However, both $ROCRD$ and $OCRD$ are incomparable with the family of *recognizable* DTDs [13]. We illustrate this result in the following example.

Example 5.4. Consider the DTD $D = \{r \to aa, a \to a|\epsilon\}$. D is not *recognizable*, but is in $OCRD$ since $\mathcal{L}(D) = \{ra^n\bar{a}^na^m\bar{a}^m\bar{r}|n, m \geq 0\} \in OCL$. Conversely, let D be the DTD $\{r \to a,\ a \to a?|b,\ b \to b?\}$. D is *recognizable* but the corresponding language, $\mathcal{L}(D) = \{ra^n b^m \bar{b}^m \bar{a}^n \bar{r} | n \geq 1, m \geq 0\}$ is not OCL [2], hence D is not in $OCRD$. Consider now the *restricted one-counter recognizable* DTD $D = \{r \to a,\ a \to a^*\}$. By straightforward testing of the necessary conditions provided in [13], one can prove that D is not *recognizable*. Finally, the DTD $D = \{r \to a_1 a_2,\ a_1 \to a_1?,\ a_2 \to a_2?\}$ is *recognizable*, but using the iteration theorem for the family of restricted one-counter languages [3] one can show that $\mathcal{L}(D) \notin ROCL$ and thus, D is *not restricted one-counter recognizable*.

Finding grammatical characterizations of *(D)ROCL* and *(D)OCL* is still an open problem. However, adapting a well known result from [3] one can infer sufficient conditions for a DTD to be in $ROCRD$. We present syntactic restrictions that yield a class of recursive DTDs that are $OCRD$.

THEOREM 5.5. *Let $D = (\Sigma, R)$ be a recursive DTD and let $D_{ECF} = (\Sigma \cup \overline{\Sigma}, N, P)$ be the associated canonical XML-grammar. Let $\{A_1, ..., A_n\} \subseteq N$ be the set of recursive nonterminals in D_{ECF} and let G be the dependency graph of D_{ECF}. Assume the following conditions are true:*

- *every node in G appears in at most one simple cycle.*
- *the regular expressions R_{A_1}, ..., R_{A_n} contain only one recursive nonterminal and that recursive nonterminal does not appear under the scope of a Kleene-star.*
- *all the nonterminals $B_i \notin \{A_1, ..., A_n\}$ have corresponding regular expressions by bottom-up substitution.*

Then the language $L(D_{ECF}) = \mathcal{L}(D)$ is a one-counter language, which means that D is in OCRD. Moreover, there is an algorithm to construct a one-counter automaton that can be used to validate an XML stream against the DTD D.

If D has only one-unambiguous content then the resulting one-counter automaton is deterministic.

Intuitively, if the dependency graph of a DTD D satisfies the conditions of *Theorem 5.5*, one can find an expression describing $\mathcal{L}(D)$ and construct based on it a one-counter automaton that precisely accepts $\mathcal{L}(D)$ [5].

EXAMPLE 5.6. Let $D = (\{r, a, b, c\}, \{r \to ab, a \to c|\epsilon, c \to a|\epsilon, b \to b|\epsilon\})$ be a recursive DTD. The corresponding canonical XML-grammar D_{ECF} has the following productions $\{ROOT \to rAB\bar{r}, A \to aC\bar{a}|\ a\bar{a}, C \to cA\bar{c}|\ c\bar{c}, B \to bB\bar{b}|b\bar{b}\}$. The language generated by the grammar is $\{(ac)^n a(\epsilon|c\bar{c})\bar{a}(\bar{c}\bar{a})^n b^m \bar{b}^m | n, m \geq 1\}$, which is not a regular language. The one-counter automaton recognizing the language generated by D_{ECF} is presented in Figure 3.

Relaxing slightly the conditions of the theorem we can find examples of the DTDs whose languages are no longer one-counter.

EXAMPLE 5.7. Let $D = (\{r, a, b\}, \{r \to a, a \to a|b|\epsilon, b \to b|\epsilon\})$ be a recursive DTD. D deviates slightly from the first two conditions of the theorem. The language generated by the corresponding canonical XML-grammar is $\{ra^n b^m \bar{b}^m \bar{a}^n \bar{r} | n \geq 1, m \geq 0\}$, which is not OCL.

EXAMPLE 5.8. Let $D = (\{r, a, b, c\}, \{r \to a, a \to ac|\epsilon, c \to b, b \to b|\epsilon\})$ be a recursive DTD. D verifies the first two conditions of the theorem but deviates slightly from the third. $\mathcal{L}(D) = \{a^n (cb^m \bar{b}^m \bar{c}\bar{a})^n | n \geq 1, m \geq 0\}$, which is not OCL.

6. CONCLUSION

This paper continues the formal investigation of the problem of on-line validation of streamed XML documents with respect to a DTD. We provided further insights on the size of the minimal (deterministic) automata that can be used for strong validation against non recursive DTDs.

Motivated by real world examples of DTDs with non-one-unambiguous content models and to capture the possible advantages of using reading windows of size greater than one, we introduced the notion of k-unambiguous regular expressions as a generalization of one-unambiguous regular expressions.

We also investigated the problem of strong validation against recursive DTDs without imposing that the streamed document be well-formed. We introduced a hierarchy of classes of DTD that can be recognized using variants of one-counter automata and provided syntactic conditions on the structure of the DTDs that ensure the existence of a one-counter automaton for performing strong validation.

A precise characterization of the DTDs recognizable by deterministic counter automata is yet not available and will be considered in future work.

7. ACKNOWLEDGMENTS

We are indebted to Leonid Libkin and Alberto Mendelzon for their guidance and feedback.

8. REFERENCES

[1] J. Berstel and L. Boasson. XML Grammars. *MFCS 2000.*

[2] J. Berstel and L. Boasson. Context-Free Languages. *Handbook of Theoretical Computer Science*, 1990.

[3] L. Boasson. Two iteration theorems for some families of languages. *J. Computer and System Sciences*, 7, 1973.

[4] A. Bruggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 140, 1998.

[5] C. Chitic and D. Rosu. On validation of XML streams using finite state machines. Technical Report CSRG-489, University of Toronto, 2004.

[6] B. Choi. What are real DTDs like. *WebDB*, 2002.

[7] D. Giammarresi, R. Montalbano, and D. Wood. Block-detereministic regular languages. *ICTCS*, 2001.

[8] V.M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16, 1961.

[9] S. Greibach. An inifinite hierarchy of context-free languages. *Journal of the ACM*, 16, 1969.

[10] C. Koch and S.Scherzinger. Attribute grammars for scalable query processing on XMl streams. *DBPL 2003.*

[11] A. Meyer P. Fisher and A. Rosenberg. Counter machines and counter languages. *Mathematical Systems Theory*, 2, 1968.

[12] A. Salomaa. Computation and Automata. *Cambridge University Press, Cambridge*, 1985.

[13] L. Segoufin and V. Vianu. Validating streaming XML documents. *PODS*, 2002.

[14] J. Paoli T. Bray and C.M. Sperberg-McQueen. XML 1.0. World Wide Web Consortium Recommendation, 1998. http://www.w3.org/TR/REC-xml/.

[15] K. Wagner and G.Wechsung. Computational Complexity. *Mathematics and its applications. VEB Deutscher Verlang der Wissenschaften, Berlin*, 1986.