Validating Streaming XML Documents

Luc Segoufin
INRIA-Rocquencourt
Luc.Segoufin@inria.fr

Victor Vianu*
U.C. San Diego
vianu@cs.ucsd.edu

ABSTRACT

This paper investigates the on-line validation of streaming XML documents with respect to a DTD, under memory constraints. We first consider validation using constant memory, formalized by a finite-state automaton (FSA). We examine two flavors of the problem, depending on whether or not the XML document is assumed to be well-formed. The main results of the paper provide conditions on the DTDs under which validation of either flavor can be done using an FSA. For DTDs that cannot be validated by an FSA, we investigate two alternatives. The first relaxes the constant memory requirement by allowing a stack bounded in the depth of the XML document, while maintaining the deterministic, one-pass requirement. The second approach consists in refining the DTD to provide additional information that allows validation by an FSA.

1. INTRODUCTION

The Extended Markup Language (XML) is emerging as the standard for data exchange on the Web. Many applications, ranging from e-commerce and B2B to scientific applications monitoring sensor or satellite data, increasingly require on-line processing of large amounts of data in XML format using limited memory. Such processing includes querying XML documents, computing running aggregates of streams of numerical data, and validating XML documents against given Document Type Definitions (DTDs).

In this paper we take a first step towards a formal investigation of processing streaming XML documents, by studying the validation question. This is an important practical problem, which is already being tackled in industry, with some commercial products developed (see related work below).

In its most restrictive form, the problem of validating streaming XML is to verify that an XML document is valid with respect to a given DTD in a single pass and using a fixed amount of memory, depending on the DTD but not on the size of the XML document. In other words, validation is done by a finite-state automaton (FSA) performing a pass on the XML document as it streams through the network, with constant memory. The problem comes in two flavors, depending on whether or not validation includes checking that the input is a well-formed XML document. Validation that includes checking well-formedness is referred to as *strong validation*. Checking satisfaction of the DTD under the assumption that the input is a well-formed XML document is referred to as simply as *validation*. It is easy to see that validation of either flavor is not possible for all DTDs using an FSA. DTDs for which (strong) validation can be done using an FSA are referred to as (strongly) recognizable DTDs.

The main results of the paper provide conditions on DTDs under which they are (strongly) recognizable. The characterization of strongly recognizable DTDs is straightforward: the DTD has to be non-recursive. Characterizing recognizable DTDs is much more intricate and technically difficult. To put the problem in perspective, note that validation with respect to a DTD amounts to checking membership of the tree associated with the XML document in a regular tree language, while validation by an FSA amounts to acceptance of the tree by a restricted form of tree-walking automaton. Thus, the connection between FSA and DTDs can be viewed as a variant (albeit simpler) of the connection between tree-walking automata and regular tree languages, a long-standing open problem [8, 9]. We obtain several kinds of results. First, we precisely characterize recognizable DTDs when the DTDs are "fully recursive", i.e. all element tags that lead to recursive tags are mutually recursive. The condition we provide can be tested in EXPTIME with respect to the size of the DTD, and in polynomial time for DTDs using 1-unambiguous regular expressions, as required by XML-Schema [4]. As a side effect, we obtain an algorithm for

*This author supported in part by the NSF under grant number IIS-9802288.

constructing from a fully recursive DTD a standard FSA that (i) always accepts only documents valid w.r.t. the DTD (but possibly more), and (ii) accepts *precisely* the documents valid w.r.t. the DTD, whenever the DTD is recognizable. The standard FSA can be constructed in time exponential in the DTD.

For DTDs that are not fully recursive, a precise characterization of recognizability remains an open question. We provide a set of necessary conditions for recognizability, as well as an extension yielding a sufficient condition. Furthermore, the construction of the standard FSA can be extended from the fully recursive case to arbitrary DTDs. It turns out that the sufficient condition is a characterization of the DTDs for which the standard FSA accepts precisely the documents valid with respect to the DTD.

For the case when validation using an FSA is not possible, we consider several alternatives. First, we relax the constant memory requirement and allow as auxiliary memory a stack of depth linear in the depth of the XML document. This is often reasonable in practice, since XML documents are typically fairly shallow, although they may be very large. We show that every DTD can be validated by a *deterministic* pushdown automaton whose stack is linear in the depth of the input document. Moreover, this holds even for DTDs extended with *specialization*, a form of element subtyping present in recent proposals such as XML-Schema. An orthogonal approach is to explore whether non-recognizable DTDs can be tweaked in reasonable ways so as to become recognizable. We show that for every DTD one can find a specialization of it which is recognizable. Intuitively, this is obtained by refining the tags of the original DTD to include more information useful for quick validation. This provides a trade-off between "accuracy" of the tags and the ability to perform efficient streaming validation.

Although limited to validation, this paper provides necessary groundwork for further investigating the problem of querying streaming XML documents. Indeed, the technical machinery developed here is likely to be useful for the more complex querying problem.

1.0.0.1 Related work

As far as we know, there is no formal work on validating or querying streaming XML. Heuristics for the evaluation of regular path queries in streaming XML documents are considered in [12]. This is part of a larger prototype, called *Tukwila*, designed for processing streaming XML documents currently developed in the University of Washington [13].

The Streaming XML Validator is a commercial product from TIBCO that performs validation of streaming XML with respect to a DTD (see http://www.tibco.com). To our understanding, their approach is based on traditional parsing techniques enhanced with heuristics geared towards streaming inputs.

A lot of work has been done on continuous queries over the Internet [6, 3] and on query subscription [16, 14]. In this scenario the query is fixed and outputs a stream of data produced on-line from an incoming stream of data. The emphasis is on filtering and incremental maintenance of views, including aggregate functions. Another large body of work focuses on numerical data streams such as sensor data.

The paper is organized as follows. Our abstraction of XML documents and DTDs, as well as basic notions on tree automata are reviewed in the Preliminaries. Section 3 concerns strongly recognizable DTDs. Section 4 presents the results on recognizable DTDs. Alternative approaches to validation are described in Section 5. Finally, brief conclusions are provided in Section 6.

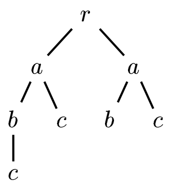
2. PRELIMINARIES

We introduce here the basic formalism used throughout the paper, including our abstraction of XML documents and DTDs. We also recall informally some basic notions related to tree automata and languages.

Let Σ be a finite alphabet.

Tree document

We abstract XML documents by "tree documents" capturing the nesting structure of elements in the document. A *tree document* t over Σ is a finite unranked tree with labels in Σ and an order on the children of each node. The following represents a simple tree document.



String representation

XML documents are a string representation of trees using opening and closing tags for each element. A streaming XML processor sees the sequence of opening and closing tags in the order in which they appear in the document. It is therefore useful to consider explicitly this string representation of an XML document. For each $a \in \Sigma$ let a itself represent the opening tag and $\bar{a}$ represent the closing tag for a. Let $\bar{\Sigma} = \{\bar{a} \mid a \in \Sigma\}$. With this notation, the string associated to the tree document above is $rabc\bar{c}\bar{b}c\bar{c}\bar{a}ab\bar{b}c\bar{c}\bar{a}\bar{r}$. More generally, we associate to each tree document t a string representation denoted $[t]$ and defined inductively as follows: if t is a single root labeled a, then $[t] = a\bar{a}$; if t consists of a root labeled a and subtrees $t_1 \ldots t_k$ then $[t] = a[t_1]\ldots[t_k]\bar{a}$. Note that Σ and $\bar{\Sigma}$ can be viewed as opening and closing multisorted parenthesis, and for each tree document t the string $[t]$ is a well-balanced string over $\Sigma \cup \bar{\Sigma}$ corresponding to a depth-first traversal of t. If T is a set of tree documents, we denote by $\mathcal{L}(T)$ the language consisting of the string representations of the tree documents in T.

Tree types and DTDs

DTDs and their variants provide a typing mechanism for XML documents. We will use several notions of types for trees. The first corresponds closely to the DTDs proposed for XML documents, and we therefore (by slight abuse) continue to use the same term. A DTD consists of an extended context-free grammar[1] over alphabet Σ (we make no distinction between terminal and non-terminal symbols). A tree document over Σ satisfies a DTD d (or is valid w.r.t. d) if it is a derivation tree of the grammar. For example, the tree document above is valid w.r.t. the DTD[2]: $\begin{array}{l} r \rightarrow a^* \\ a \rightarrow bc \\ b \rightarrow c^? \\ c \rightarrow \epsilon \end{array}$. Since regular expressions are closed under union, we can assume w.l.o.g. that each DTD has a unique rule $a \rightarrow R_a$ for each symbol $a \in \Sigma$. In the following R_a will denote both the regular expression and the corresponding regular language. The set of tree documents satisfying a DTD d is denoted by $SAT(d)$. We also denote by $\mathcal{L}(d)$ the language over $\Sigma \cup \bar{\Sigma}$ consisting of the string representations of all tree documents in $SAT(d)$, that is $\{[t] \mid t \in SAT(d)\}$. Clearly, $\mathcal{L}(d)$ is a context-free language for every DTD d. In fact, such languages of well-balanced strings of multisorted parenthesis have been studied in formal language theory under the name of *Dyck languages* [10].

The most recent DTD proposal, called XML-Schema, imposes a restriction on the regular expressions associated with each symbol: the expressions have to be 1-unambiguous. This property guarantees that the deterministic FSA for the regular expression is polynomial in the expression. Such regular expressions and other variants are studied formally in [4].

We next consider an extension of basic DTDs, also present in XML-Schema. This is motivated by a severe limitation of DTDs: their definition of the type of a given tag depends only on the tag itself and not on the context in which it occurs. For example, this means that the singleton tree document represented above cannot be described by a DTD, because the "type" of the first b differs from that of the second b. This naturally leads to an extension of DTDs with *specialization* (also called decoupled types) which, intuitively, allows defining the type of a tag by several "cases" depending on the context. Specialized DTDs have been studied in [17] and are equivalent to formalisms proposed in [2, 7]. They are present in a restricted form in XML-Schema.

Formally, we have:

[1]In an extended CFG, the right-hand sides of productions are regular expressions over the terminals and non-terminals.

[2]$b^?$ is an abbreviation for $(b|\epsilon)$.

DEFINITION 2.1. *A* specialized *DTD over Σ is a tuple $d = (\Sigma, \Sigma', d', \mu)$ where*

- *Σ and Σ' are finite alphabets;*
- *d' is a DTD over Σ'; and*
- *μ is a mapping from Σ' to Σ.*

A tree document t over Σ satisfies a specialized DTD d, if $t \in \mu(SAT(d'))$.

Intuitively, Σ' provides for some a's in Σ a set of specializations of a, namely those $a' \in \Sigma'$ for which $\mu(a') = a$. We also denote by μ the homomorphism induced on strings and trees by μ, extended whenever needed to symbols in $\bar{\Sigma}'$ by $\mu(\bar{a}') = \bar{\mu}(a')$.

Tree automata

We assume familiarity with basic notions of language theory, including (nondeterministic) finite-state automata ((N)FSA), context-free grammar (CFG) and language (CFL), and (deterministic) push-down automaton ((D)PDA) (e.g., see [11]).

We also use results on regular tree languages and tree automata. Regular tree languages are natural extensions to trees of the familiar string regular languages, and are classically defined for binary trees. A nondeterministic top-down regular tree automaton over Σ has a finite set Q of states, including a distinguished initial state q_0 and an accepting state q_f. In a computation, the automaton labels the nodes of the tree with states, according to a set of rules, called *transitions*. An internal node transition is of the form $(a, q) \rightarrow (q', q'')$, for $a \in \Sigma$. It says that, if an internal node has symbol a and is labeled by state q, then its left and right children may be labeled by q' and q'', respectively. A leaf transition is of the form $(a, q) \rightarrow q_f$ for $a \in \Sigma$. It allows changing the label of a leaf with symbol a from q to the accepting state q_f. Each computation starts by labeling the root with the start state q_0, and proceeds by labeling the nodes of the trees non-deterministically according to the transitions. The input tree is accepted if *some* computation results in labeling all leaves by q_f. A set of complete binary trees is *regular* iff it is accepted by some top-down tree automaton. Regular languages of finite binary trees are surveyed in [18]. The extension to the unranked case is discussed in [5]. Regular tree languages have similar closure properties to regular string languages, in both the ranked and unranked cases. It is worth noting that regular tree languages can be defined by many other equivalent formalisms, including bottom-up (non)deterministic automata and Monadic Second-Order logic (MSO) on the standard structures associated to trees. Interestingly, it turns out that specialized DTDs are precisely equivalent to top-down nondeterministic tree automata over unranked trees [5, 17]. Thus, they define precisely the regular tree languages.

This is more evidence that specialized DTDs are a robust and natural specification mechanism.

Another useful kind of automata on trees are the *tree-walking automata* (defined by [1] for the ranked case). These are more sequential in nature than the automata described earlier: there is a head that resides at any time at a single given node. In the unranked version, transitions depend on the current label and the state, and consist of moving the head up, down (on the left-most child), or horizontally to the left or right neighbor. It is easily seen that trees accepted by tree-walking automata can be defined in MSO, so are regular tree languages. Conversely, it is conjectured that tree-walking automata can only define a strict subset of the regular tree languages [8, 9].

3. STRONG VALIDATION OF XML DOCUMENTS

We begin with the *strong validation* problem for streaming tree documents. Recall that checking well-formedness of the XML document is now part of the validation problem. More formally, let d be a DTD (possibly specialized) over Σ and consider the associated string language $\mathcal{L}(d)$ over $\Sigma \cup \bar{\Sigma}$. We wish to characterize the DTDs d for which $\mathcal{L}(d)$ can be recognized by an FSA, i.e. $\mathcal{L}(d)$ is regular. Such DTDs are called *strongly recognizable.* We first illustrate the problem with two examples.

EXAMPLE 3.1. : Consider the DTD $d: \begin{array}{l} r \rightarrow a \\ a \rightarrow a^? \end{array}$ which defines the trees with root r containing a single branch of arbitrary length of nodes labeled a. Thus, $\mathcal{L}(d) = \{ra^n\bar{a}^n\bar{r} \mid n \in \mathbb{N}\}$ which is not regular. So, d cannot be strongly validated by an FSA and is not strongly recognizable.

EXAMPLE 3.2. : Consider the DTD $d: \begin{array}{l} r \rightarrow a^* \\ a \rightarrow b|c \end{array}$. Now $\mathcal{L}(d) = r(a(b\bar{b}|c\bar{c})\bar{a})^*\bar{r}$ which is regular. So, d is strongly recognizable.

We provide a complete characterization of the strongly recognizable (specialized) DTDs: they are precisely the non-recursive ones, defined next together with other related notions used throughout the paper.

DEFINITION 3.1. *Let d be a DTD over Σ and G_d the graph constructed as follows: its set of vertices is Σ, and for each rule $a \rightarrow R_a$ in d there is an edge from a to b for each b occurring in some word in R_a. We call G_d the* dependency graph *of d. Two labels a and b are* mutually recursive *if they belong to some cycle of G_d, and a is recursive if it is mutually recursive with itself. The DTD d is non-recursive iff G_d is acyclic. Similarly, a specialized DTD $d = (\Sigma, \Sigma', d', \mu)$ is non-recursive iff the DTD d' over Σ' is non-recursive. Finally, a DTD d is* fully recursive *if all labels from which recursive labels are reachable in G_d are mutually recursive.*

We can now show:

THEOREM 3.1. *: A specialized DTD is strongly recognizable iff it is non-recursive.*

Proof: Let $d = (\Sigma, \Sigma', d', \mu)$ be a specialized DTD. We extend μ in the obvious way in order to have $\mu(\bar{a}) = \overline{\mu(a)}$ for any $a \in \Sigma'$. Suppose first that d is strongly recognizable, i.e. $\mathcal{L}(d)$ is regular[3]. Then there exists an FSA A recognizing exactly $\mathcal{L}(d)$. Suppose towards a contradiction that d' is recursive and let $a \in \Sigma'$ be a recursive label in d'. Hence there exists a tree t in $SAT(d')$ where a repeats along one path. The string $[t]$ is of the form $ru_1av_1aw\bar{a}v_2\bar{a}u_2\bar{r}$ where u_1u_2 and v_1v_2 are well-balanced words corresponding to subtrees (or forests) of t. By iterating the recursive part of the derivation from a to a, we obtain that $[t]_n = ru_1(av_1)^naw\bar{a}(v_2\bar{a})^nu_2\bar{r}$ is also in $\mathcal{L}(d')$ for each $n > 0$. Thus, all words $\mu([t_n])$ are accepted by the FSA A. A simple pumping argument then shows that $\mu(ru_1(av_1)^{(n+k)}aw\bar{a}(v_2\bar{a})^nu_2\bar{r})$ is also accepted by A for some $k > 0$. This is a contradiction, since the string is not well-balanced.

Assume now that d is non-recursive. We can assume wlog that $\Sigma \cap \Sigma' = \emptyset$. For each $b \in \Sigma'$ construct an FSA A_b recognizing $\mu(b)R_b\overline{\mu(b)}$, where R_b is the regular expression associated to b in d'. An FSA A recognizing $\mathcal{L}(d)$ is constructed inductively as follows. Let A_0 be A_r where r is the root label. For $i \geq 0$, A_{i+1} is obtained by modifying A_i as follows. For each transition $e = (p, b, q)$ of A_i, where $b \in \Sigma'$:

1. add a copy A_e of A_b
2. add the transitions (p, ϵ, i_e) where i_e is the start state of A_e, and (f_e, ϵ, q) for each accepting state f_e of A_e
3. remove e.

Because d is non-recursive this process is sure to terminate. Note that the resulting FSA is over alphabet $\Sigma \cup \bar{\Sigma}$. It is easy to verify that the FSA recognizes $\mathcal{L}(d)$. □

To conclude the section, we consider a somewhat surprising converse to Theorem 3.1. One might legitimately wonder if there are type systems other than specialized DTDs that define families T of trees that can be strongly validated by an FSA. Interestingly, the answer turns out to be negative, as shown next.

[3]Recall that $\mathcal{L}(d)$ is a language over $\Sigma \cup \bar{\Sigma}$.

THEOREM 3.2. *: Let T be a set of trees over Σ. The language $\mathcal{L}(T)$ is regular iff there exists a non-recursive specialized DTD d such that $T = SAT(d)$.*

Proof: The "if" part follows from Theorem 3.1. For the "only if" part, suppose $\mathcal{L}(T)$ is regular so is recognized by some FSA A. From A we can easily construct a tree-walking automaton A' that performs a depth-first traversal of its input, simulating at each step the corresponding move in A and recognizing T. Since tree-walking automata define regular tree languages, and since specialized DTDs define all regular tree languages (see Preliminaries), there exists a specialized DTD d such that $T = \text{SAT}(d)$. By Theorem 3.1, d is non-recursive. □

4. VALIDATING WELL-FORMED XML DOCUMENTS

We now consider the problem of validating an XML document with respect to a given DTD d, assuming that the XML document is well formed. As before, we would like to perform the validation using an FSA. The previous requirement that $\mathcal{L}(d)$ be regular is now too strong, because the FSA only needs to work correctly on well-balanced strings representing trees. The problem can be formalized as follows. Let $\mathcal{L}(\text{Tree})$ denote the language consisting of all string representations of trees over Σ. The DTD d can be validated by an FSA iff there exists some regular language R such that $\mathcal{L}(d) = \mathcal{L}(\text{Tree}) \cap R$. Such DTDs are called *recognizable*. The characterization of recognizable DTDs turns out to be a non-trivial problem. In order to develop some intuition, we start with several examples.

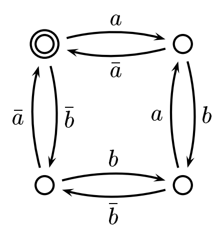
EXAMPLE 4.1. : Let us revisit the DTD d of Example 3.1: $\begin{array}{l} r \to a \\ a \to a^? \end{array}$. Recall that d is not strongly recognizable. However, it is recognizable. Indeed, if the input is known to be well balanced, it is sufficient for an FSA to check that the string is of the form $ra^*\bar{a}^*\bar{r}$. In other words, $\mathcal{L}(d) = \mathcal{L}(\text{Tree}) \cap \ ra^*\bar{a}^*\bar{r}$.

We provide two more examples of recognizable DTDs.

EXAMPLE 4.2. : Consider the DTD $\begin{array}{l} r \to a^? \\ a \to b \\ b \to a^? \end{array}$ with root r, which defines trees that are vertical alternations of a and b under root r. This DTD can be validated because $\mathcal{L}(d) = \mathcal{L}(\text{Tree}) \cap \ r(ab)^*(\bar{b}|\bar{a}|\bar{r})^*$.

EXAMPLE 4.3. : Consider the DTD $\begin{array}{l} a \to b^* \\ b \to a^* \end{array}$

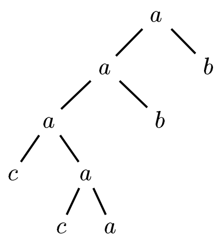
This can be validated by the following FSA that only allows the valid transitions ab, ba, $\bar{a}\bar{b}$, $\bar{b}\bar{a}$, $a\bar{a}$, $b\bar{b}$, $\bar{a}a$, $\bar{b}b$ rejecting all the others.



We next provide an example of a DTD that is *not* recognizable.

EXAMPLE 4.4. : Let d be the DTD: $\begin{array}{l} a \to (ab \mid ca \mid \epsilon) \\ b \to \epsilon \\ c \to \epsilon \end{array}$

This DTD defines trees of the form:



This DTD is not recognizable. Intuitively, even if the document is assumed to be well balanced, an FSA cannot store enough information to recall, when it reads an $\bar{a}$, whether the corresponding node had a left sibling labeled c (in which b is not allowed to its right). The formal proof follows from Lemma 4.2, see Example 4.5.

Also by way of technical warm-up, it is worth noting that conventional wisdom relating to FSA does not necessarily apply when inputs are restricted to well-balanced strings. Basic issues such as equivalence or minimization are quite different in this setting. To illustrate, consider again Example 4.2. The minimal deterministic FSA corresponding to the regular expression $r(ab)^*(\bar{b}|\bar{a}|\bar{r})^*$ has five states, and it is easily seen that this is minimal among all FSA validating the DTD. However, it is by no means unique – another deterministic FSA with five states equivalent to the first on well-balanced strings but non-isomorphic to it is the minimal one for the regular expression $r(a|b)^*(\bar{b}\bar{a}\bar{r}^*)^*$. Both FSA have the same number of states and agree on the well-balanced strings. However, the two FSA disagree on the non well-balanced words. For instance, the regular expression of Example 4.2 accepts $rab\bar{b}$ while the one above does not. Thus there is no unique minimal FSA on well-balanced inputs, unlike in the classical setting. In particular, it is not clear how to minimize an FSA validating a given recognizable DTD. However, equivalence of FSA on well-balanced inputs is decidable in EXPTIME (by a reduction to equivalence of top-down tree automata). It is open whether this can be improved.

Before proceeding, we make the following useful observation.

LEMMA 4.1. : Let T be a set of tree documents over alphabet Σ. If $\mathcal{L}(T) = \mathcal{L}(\text{Tree}) \cap R$ for some regular language R, then $T = \text{SAT}(d)$ for some specialized DTD d computable in PTIME from the FSA for R.

Proof The construction of d is similar to the classical construction of a CFG for the intersection of another CFG with a regular language, used to show closure of CFL's under intersection with regular languages [10]. The specialized alphabet consists of triples (p, a, q) where $a \in \Sigma$ and p, q are states of the FSA A_R for R. The specializations of the root r are of the form (q_0, r, q_f) where q_0 is the start state and q_f an accepting state of A_R. The regular language associated to (p, a, q) is $\{(q_1, a_1, q_2)(q_2, a_2, q_3) \ldots (q_k, a_k, q_{k+1}) \mid k > 0, a_1 \ldots a_k \in R_a, q_i$ are states of A_R, (p, a, q_1) and $(q_{k+1}, \bar{a}, q)$ are transitions in $A_R\} \cup \{\epsilon \mid \epsilon \in R_a$ and $(p, a\bar{a}, q)$ is a transition in $A_R\}$. □

We now attempt to characterize recognizable DTDs. Our basic roadmap is the following. We already know from the previous section that non-recursive DTDs are recognizable, since they are strongly recognizable. We manage to obtain a precise characterization of recognizable DTDs in the case of fully recursive DTDs. The characterization in the general case remains open. However, we make partial progress by providing necessary conditions and then extending them to sufficient conditions for recognizability. Our conjecture is that the necessary conditions we provide are actually also sufficient.

We begin with a first necessary condition in order for a DTD to be recognizable. As will be seen shortly, this condition is not sufficient in general. However, we show in Theorem 4.1 that the condition becomes sufficient in the special case of fully recursive DTDs.

LEMMA 4.2. : Let d be a recognizable DTD. Then the following hold, where α, β, u, v, w are words over Σ while x, y, z (possibly subscripted) are individual symbols:

Let k be a positive integer and x_i, z_i, $1 \leq i \leq k$ be mutually recursive symbols of d (not necessarily distinct). If $\alpha x_1 \beta \in R_{z_1}$, $\alpha' x_k \beta' \in R_{z_1}$ and $u_i x_{i-1} v_i x_i w_i \in R_{z_i}$ for $1 < i \leq k$, then $\alpha x_1 v_2 x_2 \cdots v_k x_k \beta'$ must be in R_{z_1}.

The proof of the lemma relies on a rather involved pumping argument and is sketched below. We first provide some intuition and examples. The condition relates to the inability of an FSA to enforce non-trivial horizontal constraints on the structure of trees when they concern mutually recursive symbols. This stems from the inability to remember the depth of elements, and therefore to determine when nodes are siblings. Very roughly, the rule states that what is allowed at some depth must also be allowed at any depth, modulo limited local constraints that can be enforced. More specifically, if x_1 and x_k are allowed to occur at the same level (under z_1) and x_{i-1} can be "connected" to x_i via v_i at *some* horizontal level for $1 < i \leq k$, then x_1 may be "connected" to x_k via the path $x_1 v_2 x_2 \ldots v_k x_k$ at the same level under z_1.

Remark:

Note that the condition above can be formulated as follows for $k = 1$. If x and z are mutually recursive, $\alpha x \beta \in R_z$ and $\alpha' x \beta' \in R_z$, then $\alpha x \beta'$ must also be in R_z.

We next consider a few examples.

EXAMPLE 4.5. : Recall the DTD of Example 4.4. It is not recognizable because it does not satisfy the condition in the above lemma for $k = 1$. Indeed, a is recursive in the DTD, R_a contains ab and ca, but it does not contain cab as required by Lemma 4.2.

EXAMPLE 4.6. : Consider the DTD $\begin{array}{l} a \rightarrow a \mid b \\ b \rightarrow (ab)^? \end{array}$.
This is not recognizable because it does not satisfy condition of Lemma 4.2 for $k = 2$. Indeed, a and b are mutually recursive, R_a contains a and b, R_b contains ab but R_a does not contain ab as required.

Proof of Lemma 4.2 (sketch). Suppose d is validated by an FSA A with p states. For each $a \in \Sigma$ we fix a tree $\hat{a}$ rooted at a and valid wrt d. For simplicity, when the context is clear, we also denote by $\hat{a}$ the string $[\hat{a}]$. If α is a word of $a_1 \cdots a_m$ of Σ^*, $\hat{\alpha}$ denotes the sequence of trees $\hat{a}_1 \cdots \hat{a}_m$.

We will need the following fact, whose proof is a straightforward application of the pumping lemma for regular languages.

FACT 1. *: Let A be a deterministic* FSA *over Σ, $u \in \Sigma^*$, and p the number of states of A. Let q be the state of A reached after reading u^k, $k \geq p$ starting from some state s. Then the same state q is reached after reading $u^{k+p!}$ starting from state s.*

We can assume wlog that z_1 is the root of the documents accepted by d.

The proof has two steps. We first construct a tree T in $SAT(d)$, assuming the hypothesis of the condition of the lemma. Then we modify T and obtain another tree T' that is also accepted by A and where the pattern required in the conclusion occurs under a node labeled z_1. The construction of T is somewhat tricky, as we have to ensure that a pumping-like argument can be made to show that T' is also accepted.

We start by giving some intuition for the construction. Recall that for each $a \in \Sigma$, $\hat{a}$ denotes a fixed tree rooted at a and valid for d, as well as its string representation.

We first define some "pieces" used in the construction of T. Since x_i and z_i are mutually recursive, there is a derivation in d with a path containing x_i followed by z_i and followed again by x_i. Let $\hat{x}_i^{p!}$ be the tree depicted bellow which consists of $p!$ iterations of the derivation of x_i from x_i via z_i.

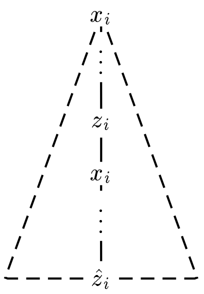


Figure 1: The trees $\hat{x}_i^{p!}$

Next, note that each z_i can be used to "connect" $\hat{x}_i^{p!}$ to $x_{i-1}^{p!}$ by expanding z_i into $\hat{u}_i x_{i-1} \hat{v}_i \hat{x}_i^{p!} \hat{w}_i$ and further expanding x_{i-1} into $\hat{x}_{i-1}^{p!}$. Also, $x_1^{p!}$ can connect to $x_k^{p!}$ by expanding z_1 into $\hat{\alpha}' x_k \hat{\beta}'$. This allows to define by induction the trees t_i, depicted in Figure 2. Let T be t_1. Thus, T is obtained by expanding t_1 with t_k, which in turn is expanded with t_{k-1}, etc. The iteration ends by expanding t_2 with $x_1^{p!}$.

Next, let T' be the tree depicted in Figure 3.

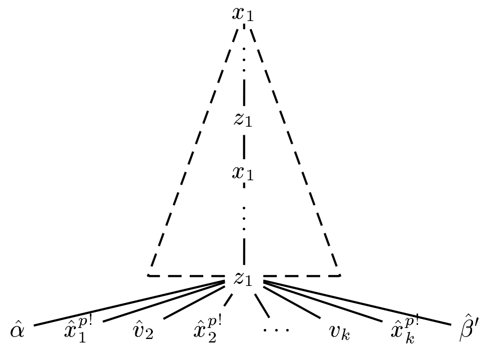


Figure 3: The tree t'

As we will prove formally, the FSA A (which has p states) cannot distinguish T from T'. The basic intuition is as follows. Consider the computation of A on T and T'. The computation can be broken down into two phases: a descending phase from the root consuming all left subtrees along a specified path in each tree, followed by an ascending phase back to the root. In T the path is the one going through the roots of the subtrees t_k. In T' it is the one going through the root of $\hat{x}_1^{p!}$. The FSA A reaches the same state after its descending phase in both T and T'. This is a consequence of Fact 1, and is shown formally below. For the ascending phase, it is enough to show that A must be in the same state after reading the substrings corresponding to $\hat{x}_i^{p!}$ in T' and t_i in T. The argument is inductive. The basis holds because the same state is reached in the descending phase. Suppose next that A is in the same state q_i after reading the substrings corresponding to $\hat{x}_{i-1}^{p!}$ in T' and t_{i-1} in T. Next, A reads $\hat{v}_i$ in both trees. This is followed in T' by $\hat{x}_i^{p!}$, and in T' by $\hat{x}_i^{p!}$ followed by an additional ascending portion to the root of t_i. However, the extra ascending string leaves the A in the same state, again as a consequence of Fact 1. This argument can be iterated to show that A returns to the root of T and T' in the same state, so T and T' are not distinguished. The formal proof is omitted. $\Box$

We next show a converse of Lemma 4.2: the necessary condition stated there in order for a DTD to be recognizable is also sufficient when the DTD is fully recursive. To do this, we first show how to construct, from any given DTD d, a standard FSA A_d that accepts all words in $\mathcal{L}(d)$ (and possibly more). We then show that for fully recursive DTD's d satisfying the conditions of Lemma 4.2, A_d accepts *precisely* the words in $\mathcal{L}(d)$.

Although we are primarily interested for the time being in fully recursive DTDs, we provide for later use the construction of A_d for arbitrary DTD's.

Construction of the standard FSA

We now outline the construction of the FSA A_d. The construction extends the simpler one involved in the proof of Theorem 3.1. Let d be an arbitrary DTD over alphabet Σ. We will use the dependency graph G_d of d. Consider the equivalence relation $\equiv$ on Σ whose equivalence classes are the strongly connected components of G_d. Let $\prec$ be the partial order on the classes of $\equiv$ where $A \prec B$ iff for some $a \in A$ and $b \in B$ there is an edge from a to b in G_d. Note that $\prec$ has a minimum element: the class of the root label. There are generally several maximal elements. We construct A_d by induction on $\prec$ starting from the maximal elements.

Let C be a maximal element of $\prec$. This means that for every $c \in C$, $R_c = \{\epsilon\}$ or words in R_c contain only symbols that are mutually recursive with c. Let A_c be an FSA corresponding to the regular expression R_c. Since A_c is non-deterministic, we can assume wlog that A_c has no "sink states", i.e. some accepting state is reachable from every state. We can also assume that the sets of states of the FSAs A_c are disjoint for different c's. Let A_C be the FSA whose set of states is the union of the sets of states of the FSAs A_c for $c \in C$. We do not need to specify at this point initial and final states

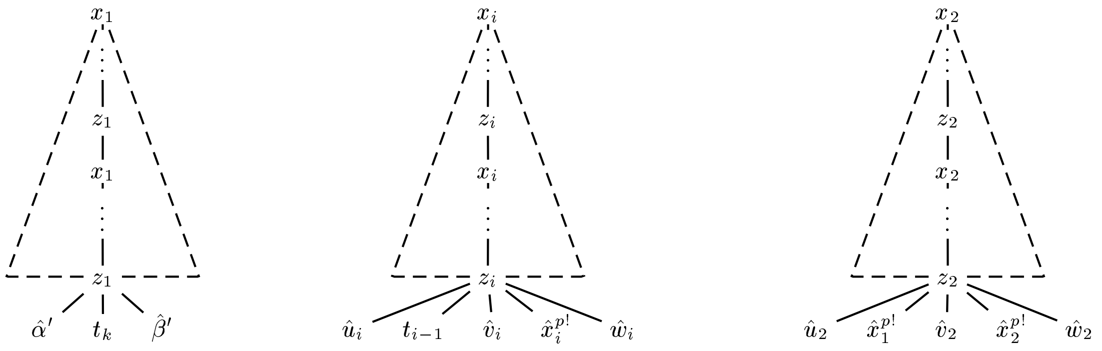


Figure 2: The trees t_i

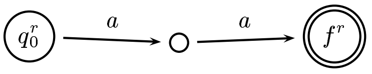
for A_C, but we mark the initial and final states of each of the participating FSAs A_c (the initial state for A_c is q_0^c and the final states f_1^c, $f_2^c, \ldots$). The transitions are defined as follows. For each transition (q, b, q') of A_c we add to A_C the transitions (q, b, q_0) and $(f, \bar{b}, q')$ for the initial state q_0^b and for each final state f_i^b of A_b.

Now suppose that C is a class of $\equiv$ for which all FSA A_D corresponding to classes D such that $C \prec D$ are already constructed. We construct A_C as follows. Again, for each $c \in C$, let A_c be an FSA corresponding to R_c (with disjoint states for distinct c's). The set of states of A_C is the union of the sets of states of the FSAs A_c for $c \in C$, similarly for the final states, and the initial state is again left unspecified. The transitions of A_C are defined as follows. As in the base case, for each $b \in C$ and transition (q, b, q') in A_c we add to A_C the transitions (q, b, q_0) and $(f, \bar{b}, q')$ for the initial state q_0 and for each final state f of A_b. Unlike the base case, we now have to take care of symbols b belonging to some class B for which $C \prec B$. For each such b we add to A_C a new disjoint copy of the already constructed A_B, together with the transitions (q, b, q_0) and $(f, \bar{b}, q')$ for the copy of the initial state q_0 and for the copies of each final state f of A_b.

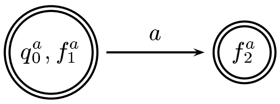
This induction allows us to construct an FSA A_C for the minimum class C containing the root label r. The final FSA A_d is obtained by adding a new start state s and final state g together with transitions (s, r, q_0) and $(f, \bar{r}, g)$ for the start state q_0 and each final state f of A_r.

We illustrate the construction of A_d with some examples.

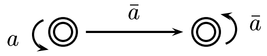
EXAMPLE 4.7. : Consider the DTD d $\begin{array}{l} r \rightarrow aa \\ a \rightarrow a^? \end{array}$ .The dependency graph G_d has the edges (r, a) and (a, a). The classes of $\equiv$ are $\{r\}, \{a\}$, and $\{r\} \prec \{a\}$. The FSA A_r and A_a are :



and



Thus the FSA associated to the equivalence class $\{a\}$ is:

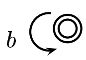


This yields the FSA A_d depicted in Figure 4.

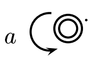
Notice that A_d recognizes all the well-balanced words of $\mathcal{L}(d)$. But it also recognizes additional well-balanced words such as $raa\bar{a}a\bar{a}\bar{a}\bar{r}$. It turns out that this is unavoidable: there is no automaton that recognizes the above DTD. This will be shown in Lemma 4.4 will show.

EXAMPLE 4.8. : Revisit now the DTD d of Example 4.3: $\begin{array}{l} a \rightarrow b^* \\ b \rightarrow a^* \end{array}$.

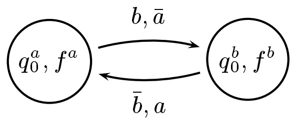
This induces one equivalence class of symbols: $\{a, b\}$. The FSA A_a and A_b are:



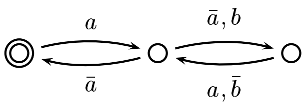
and



Thus, the FSA associated to the equivalence class $\{a, b\}$ is:



If a is assumed to be the root, this yields the FSA A_d:



Note that A_d is equivalent to the FSA of Example 4.3

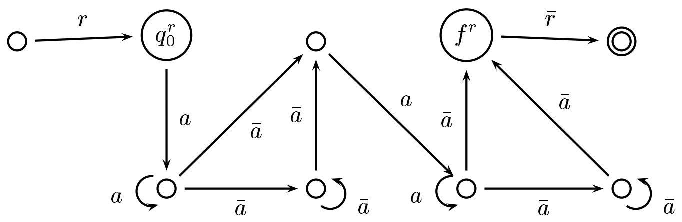


Figure 4: A_d

and the only well-balanced strings it accepts are those in $\mathcal{L}(d)$.

As expected, we can easily show, by construction, the following property of A_d.

LEMMA 4.3. : For each DTD d, let A_d be the automaton constructed above. We have:
(i) every word in $\mathcal{L}(d)$ is accepted by A_d.
(ii) A_d can be constructed from d in exponential time.

The construction of A_d in the general case takes time $O(|d|^{|\prec|})$ where $|d|$ is the maximum size of an FSA for a regular expression of d, and $|\prec|$ is the depth of the partial order $\prec$. The exponential is due to the replication of FSA's carried out in the construction.

Remark

The construction of the FSA A_d can be straightforwardly extended to specialized DTDs. Note also that A_d is non-deterministic even if the FSA's A_b associated to R_b are deterministic. The non-determinism stems from the fact that a closing tag $\bar{a}$ may lead to several states.

We can now prove a converse of Lemma 4.2, which yields a precise characterization of recognizable fully recursive DTDs.

THEOREM 4.1. *: The following are equivalent for each fully recursive DTD d:*

(i) d is recognizable,

(ii) d satisfies the conditions of Lemma 4.2, and

(iii) the set of well-balanced strings accepted by the FSA *A_d is precisely $\mathcal{L}(d)$.*

As a consequence of Theorem 4.1, we can show that it is decidable whether a fully recursive DTD d is recognizable, and therefore whether the standard associated FSA A_d can be used to validate it.

THEOREM 4.2. *: Given a specialized fully recursive DTD d over fixed alphabet Σ, it is decidable in* EXPTIME *whether d is recognizable.*

Remark:

The exponential complexity above is due to the construction of deterministic FSA for the regular expressions used by the DTD. If the DTD only uses 1-unambiguous regular expressions, such as required by XML schema [4], the complexity goes down to PTIME.

Let us now consider DTDs that are not fully recursive. Consider again the DTD of Example 4.7. Intuitively, the DTD cannot be recognized because an FSA cannot keep track of the depth in the tree and thus might allow the transition from $\bar{a}$ to a at depth different than 1. The next lemma formalizes this intuition and provides a second necessary condition for recognizability.

LEMMA 4.4. : Let d be a recognizable DTD. Then the following holds, where α, β, u, v, w are words over Σ while x, y, z (possibly subscripted) are individual symbols:

Let x_1, x_2, y, z be symbols such that x_1, x_2 and z are mutually recursive in d. If $ux_1vx_2w \in R_y$ and $u'x_1v'x_2w' \in R_z$ then $ux_1v'x_2w$ must be in R_y and $u'x_1vx_2w'$ must be in R_z.

Remark:

The condition of Lemma 4.4 can be formulated as follows in the special case when $x_1 = x_2$: (i) Suppose the occurrences of x_1 and x_2 below z are identical. Let x and z be mutually recursive in d. If $uxvxw \in R_y$ and $u'xw' \in R_z$ then $u'xw' \in R_z$ and $uxw \in R_y$. (ii) Suppose the occurrences of x_1 and x_2 below y are identical. If $uxw \in R_y$, $u'xv'xw' \in R_z$ then $uxv'xw \in R_y$ and $u'xw' \in R_z$.

EXAMPLE 4.9. : Consider the DTD of Example 4.7 $\begin{array}{l} r \rightarrow aa \\ a \rightarrow a^? \end{array}$. This DTD is not recognizable because it does not satisfy the condition of Lemma 4.4. Indeed, a is recursive, R_r contains aa, R_b contains a but R_b does not contain aa. This violates the condition of the lemma.

EXAMPLE 4.10. : Consider the DTD $\begin{array}{l} r \rightarrow ab \\ a \rightarrow b^* \\ b \rightarrow a^* \end{array}$. This DTD is not recognizable because it does not satisfy the condition of Lemma 4.4. Indeed, a and b are mutually recursive, R_r contains ab, R_b contains aa but R_r does not contain aab.

Note that, if we replace the first rule by $r \to a^*b^*$, conditions of Lemmas 4.2 and Lemma 4.4 are satisfied and the resulting DTD is recognized by A_d.

We conjecture that the necessary conditions provided by Lemmas 4.2 and 4.4 are a in fact a precise characterization for DTD recognizability. However, this remains open.

Short of a complete characterization of recognizable DTDs, we provide of characterization of when a DTD d is validated by the standard FSA A_d. The conditions are those of Lemmas 4.2 and 4.4, together with an additional condition stated next:

(*) Let α, β, u, v, w be words over Σ and x, y, z (possibly subscripted) be individual symbols. Let k and k' be positive integers. Let $(x_i)_{1 \le i \le k}$, $(z_i)_{2 \le i \le k}$, $(x'_i)_{1 \le i \le k'}$, $(z'_i)_{2 \le i \le k'}$, and y be symbols of Σ such that $x_1 = x'_1$, $x_k = x'_{k'}$, and all the x_i, x'_i, z_i, z'_i are mutually recursive in d (not necessarily distinct). If $ux_1v_1x_2 \cdots v_{k-1}x_kw \in R_y$ and for each $2 \le i \le k$ we have $\{\alpha_i x_{i-1} \beta_i, \alpha'_i x_i \beta'_i\} \subset R_{z_i}$ and for each $2 \le i \le k$ we have $u'_i x_{i-1} v'_i x_i w'_i \in R_{z'_i}$ then $ux'_1v'_1 \cdots x'_k w$ must be in R_y and, for each $2 \le i \le k$, $\alpha_i x_{i-1} v_{i-1} x_i \beta'$ must be in R_{z_i}.

The next result provides a precise characterization of the DTDs d that are validated by the standard FSA A_d.

THEOREM 4.3. : *Let d be a DTD. The following are equivalent:*

(i) d satisfies (*) and the conditions of Lemmas 4.2 and 4.4, and

(ii) the set of well-balanced strings accepted by the FSA *A_d is precisely $\mathcal{L}(d)$.*

We note that the conditions of Theorem 4.3 can be verified in time doubly exponential with respect to d. This is done by checking directly that A_d validates d, as follows. We first build a specialized DTD d' such that $SAT(d')$ consists of the trees accepted by A_d. This can be done in EXPTIME by Lemma 4.1. Next, the equivalence of d and d' can be checked in EXPTIME using a tree automata equivalence test.

To understand why the conditions in Theorem 4.3 are not a complete characterization of recognizable DTDs, consider the following example, that provides a recognizable DTD d violating (*). For this DTD, we will exhibit an FSA different from the standard A_d, that validates it.

EXAMPLE 4.11. : Consider the DTD d: $\begin{array}{l} r \to a\alpha b\beta c \\ a \to ad|\epsilon \\ d \to dc|\epsilon \\ b \to a|b|\epsilon \\ c \to b|c|\epsilon \end{array}$.

First notice that A_d does not recognize this DTD because d violates (*). Indeed the DTD satisfies the premise of (*) but not its conclusion. For example, adc is not in R_r as required. However, consider the FSA that works like A_d, but additionally counts the number of transitions $\bar{a}d$ and $\bar{d}\bar{a}$ modulo 2 and accepts only if the two are equal. It can be verified that this FSA validates d.

In summary, the conditions of Lemmas 4.2 and 4.4 are necessary in order for a DTD to be recognizable. The conditions of Theorem 4.3 are sufficient, and in particular provide a precise characterization of when the standard FSA works. The complete characterization of recognizable DTDs remains open.

5. ALTERNATIVE APPROACHES TO VALIDATION

We next consider two alternative approaches for validating DTDs that are not recognizable. The first is to relax the constant memory requirement. The second consists in refining the original DTD by adding information allowing it to be validated by an FSA.

Validation with bounded stack

We begin with relaxing the memory requirement. Specifically, we allow as auxiliary memory a stack whose depth is bounded in the depth of the XML document. The requirement that validation be done in a single, deterministic pass is maintained. This approach is appealing in practice, because many XML documents tend to be shallow even if their DTDs are recursive. We start with a simple example.

EXAMPLE 5.1. : Consider the DTD of Example 4.7 $\begin{array}{l} r \to aa \\ a \to a^? \end{array}$ which is not recognizable. However, a deterministic PDA can validate the DTD by allowing only transitions aa and $\bar{a}\bar{a}$ and remembering the current depth using the stack. In addition, the PDA allows a single transition $\bar{a}a$ and only at depth one. Note that this PDA is deterministic and its stack never exceeds the depth of the tree represented by the well-balanced input string.

Rather surprisingly, we can show that *every* specialized DTD can be strongly validated by a deterministic PDA. When the input string is well-balanced, the stack of the PDA is bounded in the depth of the tree represented by the input string.

THEOREM 5.1. : *Let d be a specialized DTD. There exists a deterministic* PDA *that accepts precisely $\mathcal{L}(d)$*

using a stack of depth bounded by the maximum number of unmatched open tags occurring as the input is read from left to right. In particular, if the input string is well-balanced, the depth of the stack is bounded by the depth of the tree represented by the input string.

Proof: Let $d = (\Sigma, \Sigma', d', \mu)$. Recall that d' is a DTD over Σ' and μ is the associated specialization mapping. We wish to check whether a string w over $\Sigma \cup \bar{\Sigma}$ represents a tree satisfying d. The stack is used to check that the string represents a tree and to keep information about the path from the root to the currently visited node in the tree. For each node along the path, the stack keeps a set of candidate specializations for the node label, compatible with the information seen so far. Intuitively, a candidate specialization a is acceptable if there are acceptable specializations of its children whose sequence forms a word in the regular language R_a associated to a by d'. The PDA must verify this recursively, and accept the input if the root is left with at least one acceptable specialization. To achieve this, the PDA simulates the run of the FSA for R_a on the children of a given node with candidate specialization a. This is done by keeping on the stack, together with each such a, the set of states reached in the FSA for R_a after reading the sequence of children seen so far, with their respective allowed specializations. This can be done because the stack symbol containing this information for a given node becomes the top of the stack every time one of its subtrees has been completely read. After reading the entire sequence of its children with their allowed specializations, a candidate specialization a for the node is discounted unless the associated set of states contains some accept state in the FSA for R_a.

We now describe the PDA in more detail. For each $a \in \Sigma'$ let A_a be the standard non-deterministic FSA for R_a, with start state q_a^0. Let Q be the disjoint union of the sets of states of the FSA's A_a. The stack alphabet of the PDA, denoted V, consists of symbols of the form (a, S) where $a \in \Sigma$, and S is a set of elements $\langle a', H\rangle$ such that $a' \in \Sigma'$, $\mu(a') = a$, and H is a subset of the states Q. Thus, V is a subset of $\Sigma \times 2^{\Sigma' \times 2^Q}$. The transitions work as follows. When $a \in \Sigma$ is read, the symbol $(a, \{\langle a', \{q_{a'}^0\}\rangle \mid a' \in \Sigma', a = \mu(a')\})$ of V is pushed on the stack. When a symbol $\bar{a} \in \bar{\Sigma}$ is read, the PDA pops the current stack symbol. If the input string is well balanced, the top of the stack must be of the form (a, S); otherwise the input is rejected. Note that, since the subtree rooted at a has been completely processed, we now know which of the candidate specializations of a are acceptable: they are the a' such that $\langle a', H\rangle \in S$ and H contains some accepting state of $A_{a'}$. At this point the new top of the stack symbol, say (b, T), needs to be updated. The symbol is popped and replaced at the top of the stack by $(b, new(T))$ where $new(T)$ contains, for each $\langle b', B'\rangle \in T$ the pair $\langle b', new(B')\rangle$ where $new(B')$ contains the states q' such that (q, a', q') is a transition of the FSA $A_{b'}$ for some $q \in B'$ and some allowed specialization a' of a occurring in S. Finally, the PDA accepts if the root node labeled r has at least one acceptable specialization r'. This information is available in the last symbol popped from the stack before it becomes empty.

It is straightforward to check that the above PDA accepts $\mathcal{L}(d)$. $\square$

Refining the DTD

We finally consider an approach to validation orthogonal to the ones examined so far. It consists of *refining* the given DTD by providing in the tags additional information that can be used for validation. The refinement is formalized by a specialization of the original DTD. More precisely, we can show the following.

THEOREM 5.2. *: For every DTD d over Σ there exists an equivalent specialized DTD $\bar{d} = (\Sigma, \Sigma', d', \mu)$ of size quadratic in d such that d' is recognizable.*

Proof: For each $a \in \Sigma$, let A_a be a standard non-deterministic FSA for the regular language R_a specified for a by the DTD d. The idea for constructing the specialized DTD $\bar{d}$ is straightforward: keep track in the tags of the children of a node a of the state of A_a in an accepting computation on the sequence of children tags. More precisely, let Q be the disjoint union of the sets of states of the FSA's A_a and let $\Sigma' = \Sigma \times Q$. The DTD d' associates to each symbol (a, q) in Σ' the regular language consisting of all words of the form $(a_1, q_1)(a_2, q_2)\ldots(a_k, q_k)$ such that $a_1 a_2 \ldots a_k \in R_a$ and (q_{i-1}, a_i, q_i) are valid transitions in A_a, $1 \le i \le k$, where q_0 is the start state and q_k an accept state for A_a. Clearly, the specialized DTD $\bar{d}$ is equivalent to d. An FSA can validate well-balanced input strings wrt the DTD d' by allowing only the following transitions:

1. $(a, q)(a_1, q_1)$ where (q_0, a_1, q_1) is a transition in A_a and q_0 is the start state of A_a;
2. $\overline{(a, q)}(b, p)$ where (q, b, p) is a transition in the FSA to which q belongs.
3. $\overline{(a, q)}\ \overline{(b, p)}$ where q is an accepting state in the FSA to which it belongs. $\square$

EXAMPLE 5.2. : Revisit the DTD of Example 4.7 $\begin{array}{l} r \to aa \\ a \to a^? \end{array}$ which is not recognizable. However the following DTD $\begin{array}{l} r \to a_1 a_2 \\ a_1 \to a_1^? \\ a_2 \to a_2^? \end{array}$ is recognizable (by the regular expression $r a_1^* \bar{a}_1^* a_2^* \bar{a}_2^* \bar{r}$) and defines a similar family of tree documents.

6. CONCLUSIONS

This paper provides a first step towards the formal investigation of processing streaming XML. We focused on the problem of on-line validation of streaming XML documents with respect to a DTD, under memory constraints. The main results provide conditions under which validation can be done in a single pass and constant memory, using an FSA. We also considered alternative approaches by relaxing the constant memory requirement or by enriching the DTD with additional information that can be used in validation.

Several questions remain open. Mainly, a precise characterization of recognizable DTDs is not yet available, except in the fully recursive case. For the general case, we conjecture that (i) the necessary conditions we provided for a DTD to be recognizable are also sufficient, and (ii) whenever a DTD d is recognizable it can be validated by the standard FSA A_d augmented with counting certain patterns modulo 2, as discussed in Example 4.11.

Another interesting open problem concerns characterizing the *specialized* DTDs that are recognizable. It can be seen that the conditions we provided for recognizable DTDs no longer work when specialization is allowed. Indeed, the problem seems considerably harder in this case. Note that, since every recognizable family of trees is necessarily definable by a specialized DTD (Lemma 4.1), characterizing the recognizable specialized DTDs would essentially close the problem of understanding which families of trees can be validated by FSA.

Finally, it would be useful to exhibit natural classes of DTDs that can always be validated by an FSA, by providing restricted specification languages for document structure that are powerful enough for a wide range of applications of practical interest.

Beyond the immediate focus on validation, we expect that the techniques developed here will also be useful in investigating the more complex problem of querying streaming XML documents.

Acknowledgment

We wish to thank Bertram Ludaescher and Yannis Papakonstantinou for pointing to us the problem of processing streaming XML. We are also grateful to Serge Abiteboul and Tova Milo for interesting discussions on the topic.

7. REFERENCES

[1] A.V. Aho and J.D. Ullman. Translations on a context free grammar. *Information and Control*, 19(19):439–475, 1971.

[2] C. Beeri and T. Milo. Schemas for integration and translation of structured and semi-structured data. In *Int'l. Conf. on Database Theory*, pages 296–313, 1999.

[3] S. Babu and J. Widom. Continuous Queries over Data Streams. In *Sigmod Record*, Sept 2001.

[4] A. Bruggemann-Klein and D. Wood. One-unambiguous regular languages. Information and Computation, 142(2):182–206, May 1998.

[5] A. Bruggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over non-ranked alphabets. Hong Kong Univ. of Science and Technology Computer Science Center Research Report HKUST-TCSC-2001-05, 2001. Available at http://www.cs.ust.hk/tcsc/RR/2001-05.ps.gz.

[6] J. Chen et al. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proc. ACM SIGMOD Conf.*, Dallas, TX, June 2000.

[7] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your mediators need data conversion! In *Proc. ACM SIGMOD Conf.*, pages 177–188, 1998.

[8] J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In J. Karhum ki, H. Maurer, G. Paun and G. Rozenberg, eds., *Jewels are forever, contributions to Theoretical Computer Science in honor of Arto Salomaa*, pp. 72-83, Springer-Verlag, 1999.

[9] J. Engelfriet, H. J. Hoogeboom and J-P. van Best. Trips on trees. *Acta Cybernetica*, 14, pp. 51-64, 1999.

[10] S. Ginsburg. The Mathematical Theory of Context-Free Languages. McGraw-Hill, 1966.

[11] J. E. Hopcroft and J. D. Ullman. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979.

[12] Z. Ives, Alon Levy and D. Weld. Efficient Evaluation of Regular Path Expressions on Streaming XML Data. Technical Report, University of Washington, 2000.

[13] Z. Ives, Alon Levy and D. Weld. Integrating Network-Bound XML Data. Data Engineering Bulletin, 24(2), 2001.

[14] Ling Liu and Calton Pu and Wei Tang and Wei Han. Conquer: A continual query system for update monitoring in the WWW. In *International Journal of Computer Systems, Science and Engineering*, 2000.

[15] F. Neven and T. Schwentick. On the Power of Tree-Walking Automata. ICALP 2000: 547-560.

[16] Benjamin Nguyen, Serge Abiteboul, Grégory Cobena, Mihai Preda. Monitoring XML data on the Web. In *Proc. ACM SIGMOD Conf.*, pp.437-448, 2001.

[17] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proc. ACM PODS*, pp. 35-46, 2000.

[18] G.Rozenberg and A. Salomaa. Handbook of Formal Language, vol. 3. Springer Verlag, 1997.