XML Validation for Context-Free Grammars

Yasuhiko Minamide[1] and Akihiko Tozawa[2]

[1] Department of Computer Science
University of Tsukuba
[2] IBM Research,
Tokyo Research Laboratory, IBM Japan, ltd.

Abstract. String expression analysis conservatively approximates the possible string values generated by a program. We consider the validation of a context-free grammar obtained by the analysis against XML schemas and develop two algorithms for deciding inclusion $L(G_1) \subseteq L(G_2)$ where G_1 is a context-free grammar and G_2 is either an XML-grammar or a regular hedge grammar. The algorithms for XML-grammars and regular hedge grammars have exponential and doubly exponential time complexity, respectively. We have incorporated the algorithms into the PHP string analyzer and validated several publicly available PHP programs against the XHTML DTD. The experiments show that both of the algorithms are efficient in practice although they have exponential complexity.

1 Introduction

String expression analysis conservatively approximates the possible string values generated by a program [CMS03b]. Minamide adopted context-free grammars as a foundation of string expression analysis and developed a string analyzer for PHP [Min05]. We consider the validation of a context-free grammar obtained by the analysis against XML schemas and develop two algorithms for deciding inclusion $L(G_1) \subseteq L(G_2)$ where G_1 is a context-free grammar and G_2 is either an XML-grammar or a regular hedge grammar, which are subclasses of context-free grammars theoretically corresponding to XML schema languages such as Document Type Definition (DTD) and RELAX NG [CM01].

To simplify the discussion on XML validation, we consider languages over a paired alphabet. Context-free languages with parentheses or paired alphabets were studied extensively in the 1960s and 1970s [McN67,Knu67,Tak75]. Let A be a base alphabet. Then, we introduce a paired alphabet consisting of two sets $\acute{A}$ and $\grave{A}$:

$$\acute{A} = \{\, \acute{a} \mid a \in A \,\} \qquad \grave{A} = \{\, \grave{a} \mid a \in A \,\}$$

where $\acute{A}$ and $\grave{A}$ correspond to the set of start tags and the set of end tags, respectively. We consider that $\acute{a}$ and $\grave{a}$ match. We write Σ for $\acute{A} \cup \grave{A}$. This notation is based on Takahashi's work on context-free grammars [Tak75].

The fundamental notion on a string over a paired alphabet is whether it is balanced. For example, $\acute{a}\acute{b}\grave{b}\acute{c}\grave{c}\grave{a}$ and $\acute{a}\grave{a}\acute{b}\grave{b}$ are balanced, but $\acute{a}\grave{b}$ and $\acute{a}\acute{b}\grave{b}$ are not.

This notion of balanced strings corresponds to well-formed documents in XML. We call the set of all balanced strings $B(\Sigma)$ the Dyck set over Σ.

As a balanced subclass of context-free languages, Berstel and Boasson proposed XML-grammars modeling DTDs and studied their formal properties [BB02]. An XML-grammar consists of a set of terminals $\Sigma = \acute{A} \cup \grave{A}$, a set of nonterminals V in one-to-one correspondence with base alphabet A, a start nonterminal S, and a set of productions. For each $a \in A$, there must be a unique production of the following form:

$$X_a \to \acute{a} R_a \grave{a}$$

where X_a is the nonterminal corresponding to a and R_a is a regular expression over V.

Example 1. Consider the following DTD, taken from [BB02].

```
<!DOCTYPE a [
   <!ELEMENT a ((a|b),(a|b)) >
   <!ELEMENT b (b)* >
]>
```

This DTD can be represented by an XML-grammar with the following productions:

$$\begin{aligned} X_a &\to \acute{a}(X_a|X_b)(X_a|X_b)\grave{a} \\ X_b &\to \acute{b}X_b^*\grave{b} \end{aligned}$$

where X_a and X_b are the nonterminals corresponding to a and b, respectively, and X_a is the start symbol.

In this formal setting, validating a context-free grammar against a DTD corresponds to checking $L(G) \subseteq L(G_{\mathsf{xml}})$ for a context-free grammar G and an XML-grammar G_{xml}. To develop an algorithm checking this inclusion, we exploit locality in DTDs and XML-grammars. They have locality in the sense that they can only describe a relation between an element and its children as can seen in the definition of XML-grammars. The algorithm has exponential time complexity and is presented in Section 4.

There is a larger class of grammars called *regular hedge grammars* corresponding to regular tree languages over unranked alphabets [Mur99]. The class of regular hedge grammars can be formulated as an extension of XML-grammars where each production has the following form:

$$X \to R$$

where R is an arbitrary regular expression over $\acute{a}Y\grave{a}$. Also there is an alternative formulation of regular hedge grammars. We obtain grammars of the same expressiveness by restricting each production to one of the following forms:

$$X \to \acute{a}Y\grave{a}Z \quad \text{or} \quad X \to \epsilon.$$

Example 2. The following is a regular hedge grammar where I is the start symbol.

$$I \to X \quad X \to (\acute{a}Y\grave{a})^*(\acute{a}X\grave{a})(\acute{a}Y\grave{a})^* \quad X \to (\acute{a}Y\grave{a})^*(\acute{b}Y\grave{b})(\acute{a}Y\grave{a})^* \quad Y \to (\acute{a}Y\grave{a})^*$$

The same language is obtained by the following productions.

$$\begin{array}{llll} I \to X & X \to \acute{a}X\grave{a}Y & X \to \acute{a}Y\grave{a}X & X \to \acute{b}Y\grave{b}Y \\ Y \to \acute{a}Y\grave{a}Y & Y \to \epsilon & & \end{array}$$

The grammar generates the set of balanced strings over $\{ \acute{a}, \grave{a}, \acute{b}, \grave{b} \}$ containing one pair of $\acute{b}$ and $\grave{b}$.

There is a regular hedge grammar that cannot be represented as an XML-grammar. The example above is indeed such a regular hedge grammar. Inversely, an XML-grammar can always be considered as a regular hedge grammar. We describe an XML validation algorithm of a context-free grammar against a regular hedge grammar in Section 3. This makes it possible to validate a context-free grammar against XML schemas such as RELAX NG which is more expressive than DTD. This validation algorithm has doubly exponential time complexity.

We introduce two new algorithms for deciding inclusion $L(G_1) \subseteq L(G_2)$ for G_1 a context-free grammar and G_2 either an XML-grammar or a regular hedge grammar. However, they do not extend known results on subclasses of context-free grammars for G_2, for which the above inclusion problem is decidable. Greibach and Friedman considered the inclusion problem for a subclass of deterministic pushdown automata called superdeterministic PDA [GF80]. They showed that it is decidable whether $L(M_1) \subseteq L(M_2)$ for M_1 an arbitrary nondeterministic PDA and M_2 a superdeterministic PDA. The complexity of their algorithm is doubly exponential in the size of the machines. It was also shown that generalized parenthesis languages studied by Takahashi [Tak75] are superdeterministic: a generalized parenthesis grammar is translated into a superdeterministic PDA, which is exponential in its size. Since regular hedge grammars are a subclass of generalized parenthesis grammars, their result is more general than ours and we can apply their algorithm to validation of a context-free grammar against a regular hedge grammar. However, if we naively estimate the complexity of the validation through a superdeterministic PDA, the complexity is triply exponential. This is one order of exponential worse than our validation algorithm.

Hereafter in this paper, we assume that a context-free grammar (CFG) is reduced. This means that every nonterminal is accessible from the start symbol and every nonterminal produces at least one terminal string.

This paper is organized as follows. In Section 2, we describe the algorithm of Berstel and Boasson, which decides whether or not every word of a context-free grammar is balanced. This is the basis of both of our algorithms. In Sections 3 and 4, we introduce our validation algorithms for regular hedge grammars and

XML-grammars, respectively. In Section 5, we describe the implementation of the algorithms as backend validators of the PHP string analyzer and show our experimental results. Finally, we review related work and present some conclusions.

2 Checking Balancedness

One of the most fundamental notions of strings over a paired-alphabet is their balancedness. Knuth [Knu67] developed an algorithm to decide whether the language of a context-free grammar is balanced for a language with a single pair of parentheses. Berstel and Boasson [BB02] extended this for a language over a paired alphabet.

Proposition 1. *Given a context-free grammar G over a paired alphabet, it is decidable whether or not its language is balanced.*

This balancedness check is the basis of validation algorithms because a grammar G is valid against some XML or regular hedge grammar only if G is balanced. However, the original algorithm by Berstel and Boasson for this balancedness check was not efficient as it could be, so that we here give an improved version of their algorithm.

Berstel and Boasson started from the following observation. We say a string ϕ is *partially balanced* if it is a factor, i.e., substring, of some balanced string. If ϕ is partially balanced, we have $a = b$ whenever $\acute{a}\psi\grave{b}$ occurs in ϕ with ψ balanced. As a result, each such ϕ is always uniquely factorized into the following form with all ϕ_i balanced.

$$\phi = \phi_1 \grave{a}_1 \phi_2 \grave{a}_2 \phi_3 \cdots \grave{a}_n \phi_{n+1} \acute{a}_{n+1} \cdots \phi_m \acute{a}_m \phi_{m+1}$$

Let us define a partial function $\rho : \Sigma^* \rightharpoonup \grave{A}^* \acute{A}^*$ by

$$\rho(\phi) = \begin{cases} \grave{a}_1 \grave{a}_2 \cdots \grave{a}_n \acute{a}_{n+1} \cdots \acute{a}_m & \phi \text{ is partially balanced} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Observe that (1) ϕ is balanced iff $\rho(\phi) = \epsilon$, and (2) $\rho(\phi\psi) = \rho(\rho(\phi)\rho(\psi))$ if ϕ and ψ are partially balanced. This means that to determine whether all strings generated from a context-free grammar G are balanced, it is sufficient to check G under interpretation by ρ.

Example 3. Consider the following grammar.

$$I \to \acute{a}\acute{a}X\grave{a}\grave{a} \qquad X \to \grave{a}\grave{a}\acute{a}\acute{a} \qquad X \to \acute{a}X\grave{a}$$

The language of this grammar is balanced. A set of strings generated from X is $\{\acute{a}^k\grave{a}\grave{a}\acute{a}\acute{a}\grave{a}^k \mid k \geq 0\}$ whose interpretation by ρ is a finite set $\{\grave{a}\grave{a}\acute{a}\acute{a}, \grave{a}\acute{a}, \epsilon\}$. We can easily see that for each string ϕ in this set $\rho(\acute{a}\acute{a}\phi\grave{a}\grave{a}) = \epsilon$.

The idea of the balancedness check is to compute the finite set $\mathrm{Irr}(X)(\subseteq \grave{A}^*\acute{A}^*) = \{\rho(\phi) \mid X \xrightarrow{*} \phi\}$ for each nonterminal X. As in the above example, given a balanced grammar this set is always finite. Furthermore, each length of $\phi \in \mathrm{Irr}(X)$ is at most exponential to the size of the balanced grammar. These facts suggest that we can stop the computation of $\mathrm{Irr}(X)$ whenever some string $\phi \in \mathrm{Irr}(X)$ is found to be longer than a given fixed length. This is the idea of Berstel and Boasson.

Let us look at this idea more precisely. In general, for each nonterminal X, we have a derivation in the form $I \xrightarrow{*} \psi X \zeta$ such that both ψ and ζ are at most of exponential length to the size of grammar [3]. Now, the balancedness implies that $\rho(\psi\phi\zeta) = \epsilon$ for any ϕ such that $X \xrightarrow{*} \phi$. We can observe that this holds iff $\rho(\psi)$, $\rho(\zeta)$ and $\rho(\phi)$ are in the following forms, $\rho(\psi) = \acute{b}_k \cdots \acute{b}_1 \acute{a}_n \cdots \acute{a}_1$, $\rho(\zeta) = \grave{c}_1 \cdots \grave{c}_m \grave{b}_1 \cdots \grave{b}_k$ ($a_n \neq c_m$), and $\rho(\phi) = \grave{a}_1 \cdots \grave{a}_n \grave{b}_1 \cdots \grave{b}_j \acute{b}_j \cdots \acute{b}_1 \acute{c}_m \cdots \acute{c}_1$ ($j \leq k$). Hence we have $|\rho(\phi)| \leq |\rho(\psi)| + |\rho(\zeta)|$.

However, this bound is not always small, as shown in the following example.

$$I \to XY_n \qquad Y_0 \to \grave{a} \qquad Y_1 \to Y_0Y_0 \qquad \ldots \qquad Y_n \to Y_{n-1}Y_{n-1}$$

We can see that $I \xrightarrow{*} X \overbrace{\grave{a} \cdots \grave{a}}^{2^n}$. Therefore, for this grammar to be balanced (e.g., define rules for $X(= X_n)$ by $X_0 \to \acute{a}, X_1 \to X_0X_0, \ldots, X_n \to X_{n-1}X_{n-1}$), each ϕ such that $X \xrightarrow{*} \phi$ should be at most 2^n in length. On the other hand, the grammar is not balanced if we define the following rules:

$$X \to \epsilon \qquad X \to X\acute{a} \qquad X \to X\acute{b}$$

where $\mathrm{Irr}(X) = \{\acute{a}, \acute{b}\}^*$. Unfortunately in checking this unbalancedness, the algorithm by Berstel and Boasson tries to compute subsets of $\mathrm{Irr}(X)$ including words at most of length 2^n, i.e., $\bigcup_{k \leq 2^n} \{\acute{a}, \acute{b}\}^k$ whose size is *doubly* exponential to the size of the grammar.

We can relax this double-exponential behavior to exponential by a small modification to the algorithm. Let $\sqsubseteq$ be the minimal ordering over $\grave{A}^*\acute{A}^*$ satisfying

$$\grave{\phi}\acute{\phi}' \sqsubseteq \grave{\phi}\grave{a}\acute{a}\acute{\phi}'.$$

Our idea is simply to compute $\mathrm{Irr}(X)$ as far as every two elements are consistent wrt this ordering, i.e., if $\phi, \phi' \in \mathrm{Irr}(X)$ then either $\phi \sqsubseteq \phi'$ or $\phi' \sqsubseteq \phi$. Again assume $I \xrightarrow{*} \psi X \zeta$ and $X \xrightarrow{*} \phi, \phi'$. By the previous discussion, if $\rho(\psi\phi\zeta) = \rho(\psi\phi'\zeta) = \epsilon$, we have both $\rho(\phi)$ and $\rho(\phi')$ in the form

$$\grave{a}_1 \cdots \grave{a}_n \grave{b}_1 \cdots \grave{b}_j \acute{b}_j \cdots \acute{b}_1 \acute{c}_m \cdots \acute{c}_1$$

[3] The depth of derivations to compute ψ and ζ is bounded by the number of nonterminals n of canonical two normal form [Har78] of G where only the productions of the following forms are allowed: $X \to YZ$, $X \to Y$, $X \to a$, and $X \to \epsilon$. Then, the sizes of ψ and ζ are at most 2^n.

```
Input CFG $(V, \Sigma, P, I)$.
Output BALANCED or NOT_BALANCED.
1 For each $X \in V$, let $bound(X) = |\rho(\psi)| + |\rho(\zeta)|$ for some $I \xrightarrow{*} \psi X \zeta$.
2 Set $\mathrm{Irr}[X] = \{\}$ for each $X \in V$
3 For each $X \to \gamma[X_1, \ldots, X_n] \in P$ where $\gamma[]$ is a context made from terminal symbols,
    and $X_1, \ldots, X_n$ are nonterminals.
      – For each tuple $\phi_1, \ldots, \phi_n$ such that each $\phi_i \in \mathrm{Irr}[X_i]$,
          • Let $\phi = \rho(\gamma[\phi_1, \ldots, \phi_n])$. If this $\phi$ is undefined, return NOT_BALANCED.
          • If $|\phi| > bound(X)$, return NOT_BALANCED.
          • If $\phi \not\sqsubseteq \phi'$ nor $\phi' \not\sqsubseteq \phi$ for some $\phi' \in \mathrm{Irr}[X]$, return NOT_BALANCED.
          • Otherwise, update $\mathrm{Irr}[X] := \mathrm{Irr}[X] \cup \{\phi\}$.
4 If some $\mathrm{Irr}[X]$ has been updated, go to 3.
5 If $\mathrm{Irr}[I] = \{\epsilon\}$ then return BALANCED, else return NOT_BALANCED.
```

Fig. 1. The algorithm of balancedness check

with only j differing. Hence $\rho(\phi)$ and $\rho(\phi')$ are always consistent wrt $\sqsubseteq$. In other words, we can stop the computation of $\mathrm{Irr}(X)$ whenever we found an inconsistent element. We obtain the algorithm in Figure 1 by extending Berstel and Boasson's algorithm with this additional consistency check.

To observe the complexity improvement, note that $\sqsubseteq$ is a linear ordering. So if $\mathrm{Irr}(X)$ only has consistent elements wrt $\sqsubseteq$, its size is bounded by the maximal length of strings in $\mathrm{Irr}(X)$.

The algorithm presented here still requires exponential time, i.e., $2^{O(n)}$-time where n is the size of the grammar. However, we conjecture that the balancedness check itself is even a PTIME problem. For this, the first step is to simplify the algorithm to check and remember only the maximal element of $\mathrm{Irr}(X)$ according to $\sqsubseteq$, rather than $\mathrm{Irr}(X)$ itself.[4] The remaining steps involve the use of a PTIME algorithm for the equivalence of straight line programs [Pla94]. We do not explain these details in this paper due to space limitation. Because the balancedness check is a subproblem of validation, and the complexity of our validation algorithms is exponential or doubly exponential, an improvement here will be canceled out in the analysis of total complexity.

3 Regular Hedge Grammar Validation

In this section, we give the first algorithm of XML validation for CFG. This algorithm determines

$$L(G) \subseteq L(G_{\mathsf{reg}})$$

where G_{reg} is specified as a regular hedge grammar. This algorithm runs in double exponential-time, i.e., time complexity bounded by $2^{2^{p(n)}}$ for some polynomial $p(n)$, to the size of inputs n.

[4] However, $\mathrm{Irr}(X)$ as we compute it here is still required in complete qualification of a grammar used in Section 4.

3.1 Regular Hedge Grammar and Binoid

We first introduce a finite algebra called binoid. We believe that a binoid is a useful algorithmic tool to solve problems related to XML and DTDs. In theory, a binoid is similar to a deterministic tree automaton whose size can grow exponentially if we construct it from a nondeterministic tree automaton. However, as we will see in the experimentation section, binoids are fairly small for practical XML schemas. For example, we can construct a finite binoid for the XHTML strict DTD with only 58 elements.

Let $B(\Sigma)$ be a Dyck set, i.e., the set of balanced strings, over Σ. We have the following proposition.

Proposition 2. *(Existence of binoid) For any regular hedge grammar G_{reg} with alphabet Σ, we have a finite algebra $\mathcal{H}(G_{\mathsf{reg}}) = (\mathcal{H}, \varepsilon, F, \hat{\_}(\_), (\_.\_))$ such that there is a (homomorphic) mapping $\_^{\circ} : B(\Sigma) \to \mathcal{H}$ such that*

(i) $\epsilon^{\circ} = \varepsilon$,
(ii) $(\acute{a}\phi\grave{a})^{\circ} = \hat{a}(\phi^{\circ})$ *for each* $a \in A$,
(iii) $(\phi\psi)^{\circ} = \phi^{\circ}.\psi^{\circ}$, *and*
(iv) $\phi \in L(G_{\mathsf{reg}})$ *iff* $\phi^{\circ} \in F$.

This algebra $\mathcal{H}(G_{\mathsf{reg}})$ is called a binoid [PQ68]. Similarly to monoids, $(\_.\_)$ is associative and ε is its unit. A difference from monoids is that we now have a new operator $\hat{\_}(\_)$, corresponding to construction of tree node, or enclosure by parentheses. We can construct a binoid from a regular hedge grammar using a variation of the algorithm of tree automata determinization [Toz06].

Example 4. A grammar in Example 2 is captured by the following binoid with three elements.

$$\begin{aligned}
&\mathcal{H} = \{\eta_0, \eta_1, \eta_\top\},\ \varepsilon = \eta_0,\ F = \{\eta_1\},\\
&\hat{a}(\eta_k) = \eta_k\\
&\hat{b}(\eta_k) = \begin{cases}\eta_1 & (k = 0)\\ \eta_\top & (\text{otherwise})\end{cases} \quad (\eta_k.\eta_{k'}) = \begin{cases}\eta_{k+k'} & (k + k' \le 1)\\ \eta_\top & (\text{otherwise})\end{cases}
\end{aligned}$$

We define the homomorphism $^{\circ}$ as $\phi^{\circ} = \eta_k$ if ϕ is a balanced string containing k occurrences, i.e., $0, 1$ or $\top$ meaning more than one, of pairs of letters $\acute{b}$ and $\grave{b}$. We can easily verify the requirements of binoid, e.g., $(\acute{b}\grave{b})^{\circ}.(\acute{a}\grave{a})^{\circ} = \eta_1.\eta_0 = \eta_1 = (\acute{b}\grave{b}\acute{a}\grave{a})^{\circ}$.

The homomorphism $(^{\circ}) \in B(\Sigma) \to \mathcal{H}$ can interpret an arabitrary balanced word as an element of $\mathcal{H}$ so that (1) constructors ϵ, vw, and $\acute{a}w\grave{a}$ for balanced words are preserved by corresponding operators ε, $(\_.\_)$ and $\hat{\_}(\_)$, and (2) the membership for $L(G_{\mathsf{reg}})$ is preserved by the membership for F. We can judge whether a set of strings is contained in $L(G_{\mathsf{reg}})$ without enumerating true strings in the set, but rather by enumerating elements of $\mathcal{H}$ computed by $\mathcal{H}$'s operators. This is the basic idea behind our algorithm.

3.2 Validation Algorithm

Assume that $G = (\Sigma, V, P, I)$ defines a balanced language. We also assume that we have a finite binoid $\mathcal{H}(G_{\mathsf{reg}}) = (\mathcal{H}, \varepsilon, F, \hat{\_}(\_), (\_.\_))$ with a homomorphism $(^\circ) \in B(\Sigma) \to \mathcal{H}$.

As mentioned, the idea of the algorithm is to interpret a set of strings generated for each nonterminal of G using the algebra $\mathcal{H}(G_{\mathsf{reg}})$. However, each string ϕ such that $X \xrightarrow{*} \phi$ is not necessarily balanced, but rather partially balanced. Therefore, we again use the factorization of ϕ. Assume that $\mathrm{Irr}(X) = \grave{a}_1\grave{a}_2 \cdots \grave{a}_n\acute{a}_{n+1} \cdots \acute{a}_m$. Each ϕ is factorized as follows:

$$\phi = \phi_1\grave{a}_1\phi_2\grave{a}_2\phi_3 \cdots \grave{a}_n\phi_{n+1}\acute{a}_{n+1} \cdots \phi_m\acute{a}_m\phi_{m+1}$$

where ϕ_i are balanced strings. Then, assume that a function ν maps a partially balanced string ϕ to $\nu(\phi) \in \mathcal{H}(\Sigma\mathcal{H})^*$ as follows.

$$\nu(\phi) = \phi_1^\circ\grave{a}_1\phi_2^\circ\grave{a}_2\phi_3^\circ \cdots \grave{a}_n\phi_{n+1}^\circ\acute{a}_{n+1} \cdots \phi_m^\circ\acute{a}_m\phi_{m+1}^\circ$$

Here $\grave{a}_1\grave{a}_2 \cdots \grave{a}_n\acute{a}_{n+1} \cdots \acute{a}_m$ is a member of $\mathrm{Irr}(X)$. Since G is balanced and $\mathcal{H}$ is finite, the set $\{\nu(\phi) \mid X \xrightarrow{*} \phi\}$ for each X is finite. Similar to the algorithm of the balancedness check, we wish to construct this set by induction.

Let us extend $\nu(\phi)$ to $\nu(\omega)$ for words $\omega \in (\Sigma \cup \mathcal{H})^*$. In the following rewrite rules, we assume $\sigma, \sigma' \in \Sigma$, $\upsilon, \omega \in (\mathcal{H} \cup \Sigma)^*$ and $\eta, \eta' \in \mathcal{H}$.

$$\begin{array}{cc}
 & \upsilon\sigma\sigma'\omega \Rightarrow \upsilon\sigma\varepsilon\sigma'\omega \\
\upsilon\eta\eta\omega \Rightarrow \upsilon(\eta.\eta')\omega & \sigma\omega \Rightarrow \varepsilon\sigma\omega \\
\upsilon\acute{a}\eta\grave{a}\omega \Rightarrow \upsilon\hat{a}(\eta)\omega & \omega\sigma \Rightarrow \omega\sigma\varepsilon \\
 & \epsilon \Rightarrow \varepsilon
\end{array}$$

Now $\nu(\omega)$ is defined as a normal form such that $\omega \Rightarrow^* \nu(\omega)$ and $\nu(\omega)$ can no longer be rewritten by $\Rightarrow$. Here, the two rules on the left interpret a given word using $\mathcal{H}$'s operators. The four rules on the right canonicalize the word by removing all leftmost, rightmost and two successive occurrences of $\sigma, \sigma' \in \Sigma$. Since the rules on the left never introduce such occurrences of σ and σ', and they decrease the length of the word, this rewrite terminates. Similar to the discussion on ρ, we can see that (1) $\phi^\circ = \nu(\phi)$ if ϕ is balanced, and hence $\nu(\phi) \in F$ iff $\phi \in L(G_{\mathsf{reg}})$, and (2) $\nu(\nu(\phi)\nu(\psi)) = \nu(\phi\psi)$ for partially balanced ϕ and ψ.

Example 5. Some examples using the binoid given in Example 4.

$$\begin{aligned}
&\nu(\grave{a}\acute{a}\grave{a}\acute{b}) = \eta_0\grave{a}(\acute{a}\grave{a})^\circ\acute{b}\eta_0 = \eta_0\grave{a}\eta_0\acute{b}\eta_0, \\
&\nu(\grave{b}\acute{a}) = \eta_0\grave{b}\eta_0\acute{a}\eta_0, \\
&\nu(\grave{a}\acute{a}\grave{a}\acute{b}\grave{b}\grave{a}) = \nu(\nu(\grave{a}\acute{a}\grave{a}\acute{b})\nu(\grave{b}\acute{a})) = \eta_0\grave{a}\eta_0.\hat{b}(\eta_0.\eta_0).\eta_0\acute{a}\eta_0 = \eta_0\grave{a}\eta_1\acute{a}\eta_0.
\end{aligned}$$

The rest of the algorithm is very close to that of the balancedness check. This is given in Figure 2.

```
Input  CFG $G = (V, \Sigma, P, I)$ defining a balanced language, and binoid $\mathcal{H}(G_{\mathsf{reg}}) = (\mathcal{H}, \varepsilon,$
    $F, \hat{\_}(\_), (\_ \cdot \_))$.
Output  VALID or INVALID.
1 Set $\mathrm{Abs}[X] = \{\}$ for each $X \in V$
2 For each $X \to \gamma[X_1, \ldots, X_n] \in P$ where $X_1, \ldots, X_n$ are nonterminals.
    – For each tuple $\omega_1, \ldots, \omega_n$ such that each $\omega_i \in \mathrm{Abs}[X_i]$,
        • update $\mathrm{Abs}[X] := \mathrm{Abs}[X] \cup \{\nu(\gamma[\omega_1, \ldots, \omega_n])\}$.
3 If some $\mathrm{Abs}[X]$ has been updated, go to 2.
4 If $\mathrm{Abs}[I] \subseteq F$ then return VALID else return INVALID.
```

Fig. 2. The validation algorithm for regular hedge grammars

3.3 Complexity

The dominant factor of the complexity is the size of $\mathrm{Abs}[X]$ for each X. The number of iterations for the outer loop of the algorithm in Fig. 2 is bounded by the number of all pairs (X, ω) such that $\omega \in \mathrm{Abs}[X]$. The inner loop for $X \to \gamma[X_1, \ldots, X_n]$ is repeated $|P|$ times, and the innermost for-each is repeated $|\mathrm{Abs}[X_1]| \times \cdots \times |\mathrm{Abs}[X_n]|$ times. Computing $\nu(\omega)$ at most requires time polynomial to $|\omega| \log |\mathcal{H}|$. The maximal length of strings in $\mathrm{Irr}(X)$ is bounded by $2^{O(|G|)}$. It is known that the size of $\mathcal{H}$ obtained from G_{reg} is at most $2^{O(|G_{\mathsf{reg}}|^2)}$. Now for each X, the size of $\mathrm{Abs}[X]$ is at most $\Sigma_{\phi \in \mathrm{Irr}(X)} |\mathcal{H}|^{|\phi|+1}$, hence $2^{2^{O(|G| + \log |G_{\mathsf{reg}}|)}}$. The O-notation absorbs all the other factors, giving $2^{2^{O(|G| + \log |G_{\mathsf{reg}}|)}}$-time total complexity of the algorithm, which is doubly-exponential to the size of G.

4 XML-Grammar Validation

We develop a validation algorithm of a context-free grammar against an XML-grammar (or DTD) by exploiting its locality. DTDs and XML-grammars have locality in the sense that they can only describe a relation between an element (tag) and its children, as we described in the introduction. This locality makes it possible to decide the inclusion problem $L(G) \subseteq L(G_{\mathsf{xml}})$ for a CFG G and an XML-grammar G_{xml} by checking local properties. As a result, we can obtain an XML-grammar validation algorithm with time complexity $2^{O(|G| + |G_{\mathsf{xml}}|)}$.

To formalize the idea, we introduce the notion of the trace and the surfaces of a balanced string by Berstel and Boasson [BB02]. Every balanced string ϕ is uniquely written into the following form:

$$\phi = \acute{a}_1 \phi_1 \grave{a}_1 \acute{a}_2 \phi_2 \grave{a}_2 \cdots \acute{a}_n \phi_n \grave{a}_n$$

where ϕ_i are balanced strings. The *trace* of a balanced string picks up only the base symbol of the toplevel tags. The trace of ϕ above is the following string.

$$\mathrm{Trace}(\phi) = a_1 a_2 \cdots a_n$$

The *surface* of a in ϕ is defined as:

$$S_a(\phi) = \{\, \mathrm{Trace}(\psi) \mid \ \acute{a}\psi\grave{a} \text{ is a substring of } \phi \text{ and } \psi \text{ is balanced} \,\}$$

This formalizes the set of sequences of tags under the a-tag in ϕ. For example, the string $\acute{a}\acute{b}\grave{b}\acute{c}\grave{c}\grave{a}\acute{a}\acute{d}\grave{d}\grave{a}$ has the following surfaces for a and b.

$$S_a(\acute{a}\acute{b}\grave{b}\acute{c}\grave{c}\grave{a}\acute{a}\acute{d}\grave{d}\grave{a}) = \{\, bc, d \,\} \qquad S_b(\acute{a}\acute{b}\grave{b}\acute{c}\grave{c}\grave{a}\acute{a}\acute{d}\grave{d}\grave{a}) = \{\, \epsilon \,\}$$

By using the surfaces of a string, we can decompose the validation of a context-free grammar G against G_{xml}. Consider the following XML-grammar as an example.

$$\begin{aligned} X_a &\to \acute{a}(X_a|X_b)(X_a|X_b)\grave{a} \\ X_b &\to \acute{b}X_b^*\grave{b} \end{aligned}$$

For the validation, it is sufficient to check the following inclusion relations for the surfaces of a and b.

$$S_a(L(G)) \subseteq L((a|b)(a|b)) \qquad S_b(L(G)) \subseteq L(b^*)$$

If we can obtain $S_a(L(G))$ and $S_b(L(G))$ as context-free grammars, the inclusion relations above are decidable since they are inclusion relations between context-free and regular languages.

We say a context-free grammar is completely balanced if $L_G(X)$ is balanced for every nonterminal of G where $L_G(X)$ is the set of strings derivable from the nonterminal X. If a context-free grammar is completely balanced, then it is balanced. The other direction does not necessarily apply.

Example 6. The following grammar is balanced, but it is not completely balanced.

$$\begin{aligned} A &\to \acute{a}\acute{b}B\grave{b}\grave{a} \\ B &\to \epsilon \mid \grave{b}\acute{b}B \end{aligned}$$

where A is a start symbol. It is not completely balanced because $B \xrightarrow{*} \grave{b}\acute{b}$, and $\grave{b}$ and $\acute{b}$ are *end* and *start* tags, respectively.

In the remainder of this section, we first show that we can compute surfaces if the grammar is completely balanced and then show that any balanced CFG can be converted into a completely balanced CFG with the same surfaces. Since the surfaces are preserved by the conversion, it can be used for validation. However, the language of the obtained completely balanced grammar may not be same as that of the original grammar.

4.1 Surfaces of a Completely Balanced CFG

We present an algorithm to obtain the surfaces of a completely balanced context-free grammar. To simplify the presentation, we restrict the format of productions of a completely balanced CFG to the following forms:

$$\begin{aligned} X &\to \acute{a}X_1 \cdots X_n\grave{a} \\ X &\to X_1 \cdots X_n \end{aligned}$$

It is easy to transform a completely balanced CFG into one with productions with these forms. From a grammar in this format, it is relatively easy to obtain the grammars representing its surfaces. The first step is to obtain the productions to produce Trace($L_G(X)$) for each nonterminal X. Each production in G is transformed as follows:

$$\begin{array}{lcl} \quad G & & \quad G' \\ X \to \acute{a} X_1 \cdots X_n \grave{a} & \Rightarrow & X \to a \\ X \to X_1 \cdots X_n & \Rightarrow & X \to X_1 \cdots X_n \end{array}$$

The first rule just picks up a since the strings derived from $X_1 \cdots X_n$ are under the start and end a tags. For example, the productions of the grammar G below are transformed as follows:

$$\begin{array}{lcl} \quad G & & \quad G' \\ A \to \acute{a}\grave{a} & \Rightarrow & A \to a \\ B \to \acute{b}\grave{b} & \Rightarrow & B \to b \\ C \to \epsilon \mid ACB & \Rightarrow & C \to \epsilon \mid ACB \\ D \to \acute{c}C\grave{c} \mid \acute{c}D\grave{c} & \Rightarrow & D \to c \mid c \end{array}$$

Then, we can construct the context-free grammar representing the surface of a in G for each $a \in A$. Consider $S_c(L(G))$ for the grammar G above. For this grammar, we have $S_c(L(G)) = \mathrm{Trace}(C) \cup \mathrm{Trace}(D)$ because a pair of $\acute{c}$ and $\grave{c}$ occurs only in the following two productions.

$$D \to \acute{c}C\grave{c} \qquad D \to \acute{c}D\grave{c}$$

Therefore, $S_c(L(G))$ can be represented with a grammar with the following productions:

$$\begin{array}{ll} A \to a & D \to c \\ B \to b & I \to C \mid D \\ C \to \epsilon \mid ACB & \end{array}$$

where I is the start symbol. This grammar generates the following language:

$$S_c(L(G)) = \{\, a^n b^n \mid n \geq 0 \,\} \cup \{\, c \,\}$$

The context-free grammars for $S_a(L(G))$ and $S_b(L(G))$ are constructed in the same manner. Then, we can validate G against an XML-grammar using the surfaces.

4.2 Transformation into a Completely Balanced CFG

The rest of our validation algorithm is to transform a balanced CFG into a completely balanced CFG with the same surfaces. This is the most involved part of the XML-grammar validation algorithm. The following grammar shows that there is a balanced CFG that cannot be represented with a completely balanced CFG [Knu67].

$$I \to A\grave{a} \qquad A \to \acute{a} \mid \acute{b}\grave{b}A\acute{c}\grave{c}$$

This grammar generates $\{\, (\acute{b}\grave{b})^n \acute{a} (\acute{c}\grave{c})^n \grave{a} \mid n \geq 0 \,\}$.

We say that a context-free grammar is completely qualified if $\mathrm{Irr}(X)$ is a singleton for every nonterminal X. Given a balanced CFG G, we can construct a completely qualified CFG G' where $L(G') = L(G)$ [Knu67]. Therefore, in this section, we assume a balanced CFG is completely qualified and write $\mathrm{Irr}(X) = \phi$ if $\mathrm{Irr}(X) = \{\, \phi \,\}$.

The transformation we introduce is based on the factorization of partially balanced strings. Consider the following factorization of a partially balanced string ϕ:

$$\phi \equiv \phi_1 \grave{a}_1 \phi_2 \grave{a}_2 \phi_3 \cdots \phi_m \grave{a}_m \phi_{m+1} \acute{a}_{m+1} \phi_{m+2} \cdots \phi_n \acute{a}_n \phi_{n+1}$$

where ϕ_i are balanced. We define the i-th factor $F_i(\phi)$ of ϕ as $F_i(\phi) = \phi_i$.

Let $G = (V, \Sigma, P, I)$ be a balanced CFG. We construct a completely balanced CFG G' with the same surfaces as follows. For each nonterminal X, we introduce nonterminals X_i $(1 \leq i \leq |\mathrm{Irr}(X)| + 1)$. Let V' be the set of nonterminals X_i introduced above. Then, we define a function F from V to $V'(\Sigma V')^*$ as follows:

$$F(X) = X_1 \grave{a}_1 X_2 \grave{a}_1 X_3 \cdots X_m \grave{a}_m X_{m+1} \acute{a}_{m+1} X_{m+2} \cdots X_n \acute{a}_n X_{n+1}$$

where $\mathrm{Irr}(X) = \grave{a}_1 \grave{a}_2 \cdots \grave{a}_m \acute{a}_{m+1} \cdots \acute{a}_n$. This function F on nonterminals is naturally extended to a function on $(\Sigma \cup V)^*$ by $F(\acute{a}) = \acute{a}$ and $F(\grave{a}) = \grave{a}$ for all base symbol a.

With this function we can expand production $X \to \gamma$ of G as follows.

$$F(X) \to F(\gamma)$$

By construction, $F(\gamma)$ must have the following form:

$$F(\gamma) = \gamma_1 \grave{a}_1 \gamma_2 \grave{a}_2 \gamma_3 \cdots \gamma_m \grave{a}_m \gamma_{m+1} \acute{a}_{m+1} \gamma_{m+2} \cdots \gamma_n \acute{a}_n \gamma_{n+1}$$

where γ_i are balanced by considering that nonterminals are balanced. This is because G is completely qualified and thus $\mathrm{Irr}(\gamma) = \mathrm{Irr}(X)$. Then, we construct $G' = (V', \Sigma, P', I_1)$ where P' contains the following productions for each production $X \to \gamma \in P$.

$$X_i \to F_i(F(\gamma)) \quad (1 \leq i \leq |\mathrm{Irr}(X)| + 1)$$

It is clear that only balanced strings can be derived from each nonterminal X_i in G' since the right-hand side of each production is a balanced factor. Therefore, G' is a completely balanced CFG and the following are satisfied:

$$L(G) \subseteq L(G')$$

$$S_a(G) = S_a(G') \quad \text{for each } a \in A$$

We proved these properties for a CFG G in Chomsky normal form. The first property is easily shown by construction and the second is obtained from the following property:

$$\mathrm{Trace}(L_{G'}(X_i)) = \mathrm{Trace}(F_i(L_G(X)))$$

An expanded production $F(X) \to F(\gamma)$ above can be considered as a context-sensitive production rule. To obtain a CFG, we split it into several productions for factors. This is the source of approximation in the transformation.

Example 7. Consider again the following grammar considered by Knuth:

$$I \to A\grave{a} \qquad A \to \acute{a} \mid \acute{b}\grave{b}A\acute{c}\grave{c}$$

where $\mathrm{Irr}(A) = \acute{a}$ and $\mathrm{Irr}(I) = \epsilon$. This grammar generates $\{(\acute{b}\grave{b})^n\acute{a}(\acute{c}\grave{c})^n\grave{a} \mid n \geq 0\}$. We have $F(A) = A_1\acute{a}A_2$ and $F(I) = I_1$. Therefore, the productions for A are expanded as follows:

$$A_1\acute{a}A_2 \to \acute{a} \mid \acute{b}\grave{b}A_1\acute{a}A_2\acute{c}\grave{c}$$

From $F_1(\acute{b}\grave{b}A_1\acute{a}A_2\acute{c}\grave{c}) = \acute{b}\grave{b}A_1$ and $F_2(\acute{b}\grave{b}A_1\acute{a}A_2\acute{c}\grave{c}) = A_2\acute{c}\grave{c}$, we obtain the following grammar:

$$I_1 \to A_1\acute{a}A_2\grave{a} \qquad A_1 \to \epsilon \mid \acute{b}\grave{b}A_1 \qquad A_2 \to \epsilon \mid A_2\acute{c}\grave{c}$$

This grammar generates $\{\ (\acute{b}\grave{b})^n\acute{a}(\acute{c}\grave{c})^m\grave{a} \mid n, m \geq 0\ \}$. The constraint between n and m is lost by the transformation and consequently they do not have the same language. However, it is clear that it has the same surfaces as those of the original grammar.

4.3 Complexity

Let n and m be the sizes of a balanced CFG G and an XML-grammar G_{xml}, respectively. We assume that the length of the right-hand side of a production of G is at most two. As described in Section 2, both the cardinality of $\mathrm{Irr}(X)$ and the maximal length of strings in $\mathrm{Irr}(X)$ are bounded by 2^n for every nonterminal X in G. We define $\iota(G)$ for a balanced CFG G as follows:

$$\iota(G) = max\{\ |\gamma| \mid \gamma \in \mathrm{Irr}(X) \text{ for some nonterminal } X \text{ in } G\ \}$$

The first step of the validation algorithm is to obtain an equivalent completely qualified CFG G_1. We can obtain G_1 with at most 2^{2n} productions by the transformation of Knuth [Knu67] in time $2^{O(n)}$. The length of the right-hand side of a production in G_1 is again at most two and $\iota(G) = \iota(G_1)$.

The second step is to obtain a completely balanced CFG G_2 that has the same surfaces with G_1. Each nonterminal X with $\mathrm{Irr}(X) = \grave{a}_1\grave{a}_2 \cdots \grave{a}_m\acute{a}_{m+1} \cdots \acute{a}_n$ in G_1 is translated as follows.

$$F(X) = X_1\grave{a}_1X_2\grave{a}_1X_3 \cdots X_m\grave{a}_mX_{m+1}\acute{a}_{m+1}X_{m+2} \cdots X_n\acute{a}_nX_{n+1}$$

We have $|F(X)| = 2|\mathrm{Irr}(X)| + 1 \leq 2\iota(G_1) + 1$. A production $X \to \gamma$ in G_1 is translated into the following productions.

$$X_i \to F_i(F(\gamma)) \quad (1 \leq i \leq |\mathrm{Irr}(X)| + 1)$$

programs	size		the numbers of		time(sec)	
	depth	Irr(X)	nonterminals	productions	binoid	surface
WebCalendar	8	4	102	170	0.0121	0.0492
marktree	∞	4	32	54	0.0023	0.0127
phpScheduleIt	15	8	24	42	0.0062	0.0191
mrtask	11	6	50	70	0.0059	0.0281

Table 1. XHTML validation

It is observed that each production rule is translated into at most $\iota(G_1) + 1$ production rules and $|F(\gamma)| \leq 2(2\iota(G_1) + 1)$ since $|\gamma| \leq 2$. Thus, the size of G_2 is bounded by $O(2^{2n} \cdot \iota(G_1)) = O(2^{3n})$ and it is obtained in time $2^{O(n)}$.

To validate a context-free grammar G_2 against an XML-grammar G_{xml}, we need to check $S_a(G_2) \subseteq L(R_a)$ for each $X_a \to \acute{a} R_a \grave{a}$ in G_{xml}. The size of the deterministic automaton for R_a has at most 2^m states. Because intersection emptiness between a context-free grammar and an automaton can be checked in cubic time, the inclusion relation for each surface can be checked in time $2^{O(n+m)}$. Thus, the complexity of the algorithm in total is in $2^{O(n+m)}$.

5 Experimental Results

We have implemented our validation algorithms as backend validators of the PHP string analyzer developed by Minamide [Min05]. The analyzer generates a CFG that conservatively approximates the string output of a PHP program. It is available from `http://www.score.cs.tsukuba.ac.jp/~minamide/phpsa/`. In our experiments, we checked the validity of Web pages generated by a PHP program against the XHTML specification. However, we ignored attributes in our experiments and only checked the constraints on elements imposed by the specification. As a preliminary step of validation, the analyzer eliminates comments, attributes, and non-tag texts from a CFG and obtains the corresponding CFG over a paired alphabet. The transformations for this simplification are implemented as string transducers.

In our implementation of the validation algorithms, we first extract the set of element names appearing in a CFG obtained by the analyzer and delete the elements from the DTD that do not appear in the set. Without this optimization, it takes approximately 0.2 seconds to construct the binoid for the XHTML DTD and this dominates validation time in the binoid-based validation.

We applied our validation algorithms to several PHP programs available from SourceForge and validated the top Web pages generated by them. We repaired several validity bugs in these programs and had to modify the programs to improve the precision of the analysis. The experiments were performed on a Linux PC with two Opteron processors (2.8 GHz) and 8 GB memory. The CFGs were validated against XHTML version 1.0 Transitional DTD[5]. Table 1 summarizes our experiments. The first four columns show the various information concerning the grammars over a paired-alphabet obtained by the analyzer. We checked the grammars and found that all the grammars are completely qualified. The columns 'depth' and '$\mathrm{Irr}(X)$' show the maximum nesting depth of elements (tags) and the maximum size of $\mathrm{Irr}(X)$. In the last two columns, the table shows the validation time for the binoid-based and the surface-based validation algorithms[6]. These do not include the time spent in obtaining the CFGs. These results show that both algorithms are fast enough for common server-side programs even if they have theoretically exponential complexity. We think that it is because the size of $\mathrm{Irr}(X)$ is small in practice, as shown in the table, and a regular expression in a content model of DTD must be deterministic.

It is interesting that the binoid-based validation is faster in these experiments although it has higher complexity. This may be because the implementation of the binoid-based validation is simpler than that of the surface-based validation. However, it is also straightforward to write an artificial program where both of the algorithms show their exponential behavior. Consider the following program where `$x = $x.$x; $y = $y.$y;` is repeated n times.

```
$x = "<div>"; $y = "</div>";
if (rand()) $x = $x."<p></p>";
$x = $x.$x; $y = $y.$y;
...
$x = $x.$x; $y = $y.$y;
echo $x; echo $y;
```

The grammar obtained for this program is completely qualified and the size of $\mathrm{Irr}(\cdot)$ for the variables `$x` and `$y` is in $O(2^n)$. The surface-based algorithm shows exponential behavior for this program and it takes 1.0, 6.5, and 34.5 seconds to validate it for $n = 10, 11, 12$, respectively. On the other hand, the binoid-based algorithm shows doubly exponential behavior for this program because of the `if`-statement in the program and can validate it only when $n \leq 5$.

6 Related Work

The PHP string analyzer originally supported only the inclusion checking between a CFG and a regular expression [Min05]. This checking can partially support validation of dynamically generated Web pages by restring their depth. It is because the set of valid Web pages can be described with a regular language if we restrict their depth. The algorithms in this paper give more direct and general solutions to the problem.

[5] The content model $(r)+$ was interpreted as $(r)*$ in the experiments to circumvent imprecision due to analysis of loops in a program.

[6] We measured the time spent to validate a CFG 100 times. The table shows an average time calculated from the total time.

The XML validation algorithms presented in this paper depend on previous work on context-free grammars over languages with parentheses. A parenthesis grammar is a context-free grammar over a language with a single pair of parentheses where each production has the form of $A \to (\theta)$ where θ does not contain parentheses. McNaughtotn showed that equivalence of parenthesis grammars is decidable [McN67]. Knuth extended the result and showed that there exists an algorithm to determine whether a context-free grammar is a parenthesis language [Knu67]. Berstel and Boasson extended the theory of context-free grammars over languages with parentheses to study the language described by a DTD [BB02]. In this paper, we have developed a surface-based validation algorithm by exploiting their results that the language of an XML-grammar has locality and can be characterized by its surfaces, and the regular hedge grammar validation based on their algorithm for checking balancedness of a context-free language.

Extensive studies have been done in tree-based validation of dynamically generated HTML/XML documents [CMS03a,HP03,HVP05]. The motivation of the work is the same as ours, but the validation algorithms developed in these works cannot be directly applied to our setting of string-based validation. As shown in Section 4, we can retrieve a tree-based language for a balanced CFG with the approximation preserving surfaces of the language. Thus, after the transformation, the methods for tree-based validation can be applied in principle. Brabrand, Møller, and Schwartzbach proposed summary graphs to approximate the set of dynamically generated XHTML documents [BMS01]. Although summary graphs can express constraints on attributes, they basically correspond to *completely* balanced CFGs. The validation of a completely balanced CFG can be considered as a variant of their validation algorithm for summary graphs.

7 Conclusion

We have presented two new algorithms validating a context-free grammar against a regular hedge grammar and an XML-grammar. Although both have exponential complexity, it is shown that they are efficient in practice. Our validation algorithms for regular hedge grammars and XML-grammars have doubly exponential and exponential time complexity. We plan to establish the lower bounds for these validation problems. For simpler problems, it is known that the inclusion problems for regular expressions and regular hedge grammars are PSPACE-complete and EXPTIME-complete, respectively. However, gaps remain between the results and the complexity of our algorithms.

We have considered validation against a subclass of balanced context-free grammars, such as XML-grammars and regular hedge grammars, but legacy server-side programs generate HTML, which is not in XML format. In order to validate those Web pages, we need to consider validation against a grammar that has an unbalanced language. Although the inclusion problem between two context-free grammars is undecidable in general, we think that it is possible to validate a context-free grammar against the HTML specification because it is designed to be unambiguous and where end tags are omitted can be determined.

References

[BB02] Jean Berstel and Luc Boasson. Formal properties of XML grammars and languages. *Acta Informatica*, 38(9):649–671, 2002.

[BMS01] Claus Brabrand, Anders Møller, and Michael I. Schwartzbach. Static validation of dynamically generated HTML. In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis For Software Tools and Engineering*, pages 38–45, 2001.

[CM01] J. Clark and M. Murata. RELAX NG specification, 2001. http://www.oasis-open.org/committees/relax-ng/spec.

[CMS03a] Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach. Extending Java for high-level web service construction. *ACM Transactions on Programming Languages and Systems*, 25(6):814–875, 2003.

[CMS03b] Aske Simon Christensen, Andres Møller, and Michael I. Schwartzbach. Precise analysis of string expressions. In *Proceedings of the Static Analysis Symposium (SAS)*, volume 2694 of *LNCS*, pages 1–18, 2003.

[GF80] Sheila A. Greibach and Emily P. Friedman. Superdeterministic PDAs: A subcase with a decidable inclusion problem. *Journal of the Association for Computing Machinery*, 27(4):675–700, 1980.

[Har78] Michael A. Harrison. *Introduction to Formal Language Theory*, chapter 4. Addison-Wesley, 1978.

[HP03] Haruo Hosoya and Benjamin Pierce. XDuce: A statically typed XML processing language. *ACM Transactions on Internet Technology*, 3(2):117–148, 2003.

[HVP05] Haruo Hosoya, Jérôme Vouillon, and Benjamin Pierce. Regular expression types for XML. *ACM Transactions on Programming Languages and Systems*, 27(1):46–90, 2005.

[Knu67] Donald E. Knuth. A characterization of parenthesis languages. *Information and Control*, 11(3):269–289, 1967.

[McN67] Robert McNaughton. Parenthesis grammars. *Journal of the Association for Computing Machinery*, 14(3):490–500, 1967.

[Min05] Yasuhiko Minamide. Static approximation of dynamically generated Web pages. In *Proceedings of the 14th International World Wide Web Conference*, pages 432–441. ACM Press, 2005.

[Mur99] Makoto Murata. Hedge automata: a formal model for XML schemata, 1999. `http://www.xml.gr.jp/relax/hedge_nice.html`.

[Pla94] Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In *Algorithms – ESA '94 (Utrecht)*, volume 855 of *LNCS*, pages 460–470. Springer, 1994.

[PQ68] C. Pair and A. Quere. Définition et étude des bilangages réguliers. *Information and Control*, 13(6):565–593, Dec 1968.

[Tak75] Masako Takahashi. Generalizations of regular sets and their application to a study of context-free languages. *Information and Control*, 21(1):1–36, 1975.

[Toz06] Akihiko Tozawa. XML type checking using high-level tree transducer. In *Functional and Logic Programming, 8th International Symposium, FLOPS 2006*, pages 81–96, 2006.