A Translation from the HTML DTD into a Regular Hedge Grammar

Takuya Nishiyama and Yasuhiko Minamide

Department of Computer Science, University of Tsukuba
{nishiyama,minamide}@score.cs.tsukuba.ac.jp

Abstract. The PHP string analyzer developed by the second author approximates the string output of a program with a context-free grammar. By developing a procedure to decide inclusion between context-free and regular hedge languages, Minamide and Tozawa applied this analysis to checking the validity of dynamically generated XHTML documents. In this paper, we consider the problem of checking the validity of dynamically generated HTML documents instead of XHTML documents.

HTML is not specified by an XML schema language, but by an SGML DTD, and we can omit several kinds of tags in HTML documents. We formalize a subclass of SGML DTDs and develop a translation into regular hedge grammars. Thus we can validate dynamically generated HTML documents. We have implemented this translation and incorporated it in the PHP string analyzer. The experimental results show that the validation through this translation works well in practice.

1 Introduction

The PHP string analyzer was developed by the second author to check various properties of PHP programs [Min05]. It approximates the string output of a program with a context-free grammar. Minamide and Tozawa applied the analysis to checking the validity of XHTML documents generated dynamically by a server-side program [MT06]. They developed a decision procedure that checks inclusion between a context-free and regular hedge languages. The validity is checked by applying the procedure to the context-free and regular hedge grammars obtained from a program and the XHTML DTD.

In this paper, we consider the problem of checking the validity of dynamically generated HTML documents instead of XHTML documents. HTML is not based on XML, but on SGML [Gol90], and its specification is given as an SGML DTD. Unlike XML documents, we can omit several kinds of tags in HTML documents according to the HTML DTD [Wor99]. Models of XML schema languages have been studied based on the theory of formal languages. Murata proposed a regular hedge grammar as a foundation of XML schemas [Mur99], and XML DTDs were modeled as a subclass of context-free grammars called XML-grammars by Berstel and Boasson [BB02]. However, the presence of tag omission makes it harder to model an SGML DTD as a formal language. As far as we know, there is no formal model of SGML DTD.

O.H. Ibarra and B. Ravikumar (Eds.): CIAA 2008, LNCS 5148, pp. 122–131, 2008.

In this paper, we formalize a subclass of SGML DTDs and develop a translation from the subclass into regular hedge grammars. The subclass is expressive enough to include the HTML DTD, making it possible to validate dynamically generated HTML documents based on the decision procedure of Minamide and Tozawa.

Even though it seems rather difficult to formalize general SGML DTDs and represent them with regular hedge grammars, we found that the following two properties hold for the HTML DTD. (a) It is possible to check whether an end tag can be omitted or not by looking at the next element of an SGML document, and (b) the number of direct nestings of tag omissions is bounded by a fixed number determined by the DTD.

We can write an HTML document like the following.

```
<p> <p> <p> ...
```

It seems that `p` elements can be nested any number of times. However, according to the HTML DTD, a `p` element cannot appear as a child of `p`. Thus, the string above is interpreted as the following string by inserting `</p>` symbols.

```
<p> </p><p> </p><p> </p> ...
```

Thus, `p` elements are not nested in the string.

Based on these observations, we first formalize the language of an SGML DTD satisfying the first condition as an image of a regular hedge language under a transducer, omitting end tags from a valid string not including tag omissions. Furthermore, we show that if an SGML DTD satisfies both conditions, the image is also a regular hedge language, and we develop a translation from the subclass of SGML DTDs into regular hedge grammars. We have implemented this translation and incorporated it in the PHP string analyzer. The experimental results show that the validation through the translation works well in practice.

This paper is organized as follows. In Section 2, we review SGML and SGML DTD. We formalize the subset of SGML DTDs by modeling end-tag omission with a transducer in Section 3. Section 4 introduces a further restricted class of SGML DTD and gives a translation from a DTD into a regular hedge grammar. The translation is extended to support the exclusion feature of SGML DTD in Section 5. Finally, we show our experimental results.

2 SGML and SGML DTD

SGML is a predecessor of XML, and a markup language for structured documents such as XML. One of the essential differences between SGML and XML is that we can omit tags in SGML documents. In SGML, a document type is specified with the schema language DTD. Although it is quite similar to the DTD for XML, it has more features to specify the type of documents: tag omission, inclusion, and exclusion are the features used in the HTML DTD that are specific to SGML DTD.

Let us consider the following SGML DTD.

```
<!DOCTYPE friends [
 <!ELEMENT friends O O (person)* >
 <!ELEMENT person  - O (name, phone?) >
 <!ELEMENT name    - O (#PCDATA) >
 <!ELEMENT phone   - O (#PCDATA) >
] >
```

This declaration specifies content-models of five elements. For example, inside the `friends` element any sequence of `person` elements can appear, as specified by the regular expression `(person)*`.

This declaration also specifies tag omission for each element, namely whether or not the start and end tags of the element are optional, by two characters between the element name and the content-model. If the first character is "`O`", then the start tag of the element can be omitted. Otherwise, it is mandatory. The second character indicates the same for the end tag of the element. For the DTD above, only the start tag of the element `friends` and the end tags of the elements `friends`, `person`, `name`, and `phone` can be omitted. We call elements with optional end tags end-tag omittable elements.

However, tags cannot be omitted everywhere even if it is specified so in a DTD. A tag in a document can only be omitted if the structure of the document can be uniquely determined without the tag. Warmer and van Egmond clarified the condition in the specification as follows [WvE89]. An end tag can be omitted only if it is followed by the end tag of another open element, or by the start tag of another element or an SGML character that is not allowed in the element's content-model.

Let us clarify the condition with the following example.

```
<!DOCTYPE a [
 <!ELEMENT a - - (b|c|d)* >
 <!ELEMENT b - O (c,d)+ >
 <!ELEMENT c - - (#PCDATA) >
 <!ELEMENT d - - (#PCDATA) >
] >
```

The end tag `</b>` is specified to be omittable in this DTD and the following document is valid.

```
<a><b><c>A</c><d>B</d><c>C</c></a>
```

It is interpreted as the following by inserting `</b>` before the second `<c>`. This is because `c,d,c` is not allowed in the content model `(c,d)+` of the element `b`.

```
<a><b><c>A</c><d>B</d></b><c>C</c></a>
```

On the other hand, `</b>` in the following document cannot be omitted because `c,d,c,d` is allowed in `b`.

```
<a><b><c>A</c><d>B</d></b><c>C</c><d>D</d></a>
```

As this example shows, to decide whether an end tag can be omitted, it is necessary to look ahead to an unbounded number of elements in general.

3 Formalization of SGML DTD

As stated above, an SGML specific feature that is often used in HTML documents is the omission of end tags. Although start tags of several elements including `HTML`, `HEAD`, and `BODY` are specified as optional in the HTML DTD, it is rather rare to omit them (at least in a server-side program). Thus, in this paper, we formalize a subset of SGML DTDs that allow omission only of end tags. Furthermore, to simplify our formalization, we introduce a subclass of SGML DTDs that makes it possible to check whether an end tag can be omitted or not by looking only at the next element of an SGML document. The class is expressive enough to deal with the HTML DTD as we discussed in the introduction.

To formalize SGML documents and DTDs, we consider SGML documents as strings over a paired alphabet. Let Σ be a base alphabet corresponding to the set of element names. We introduce a paired alphabet consisting of two sets $\acute{\Sigma}$ and $\grave{\Sigma}$:

$$\acute{\Sigma} = \{\acute{a} \mid a \in \Sigma\} \qquad \grave{\Sigma} = \{\grave{a} \mid a \in \Sigma\}$$

where $\acute{\Sigma}$ and $\grave{\Sigma}$ correspond to the set of start tags and the set of end tags, respectively. For example, the following document is described as $\acute{a}\acute{b}\acute{c}\grave{c}\grave{b}\grave{b}\grave{a}$.

```
<a><b><c></c></b><b></b></a>
```

If we omit the end tag of `c`, we obtain an unbalanced string, $\acute{a}\acute{b}\acute{c}\grave{b}\grave{b}\grave{a}$. We discuss the validity of SGML documents based on this representation, and thus ignore the attributes and the text parts of a document.

Definition 1. *We formalize SGML DTD as a 4-tuple* $D = (\Sigma_e, \Sigma_m, M, r)$.

- Σ_e *is the finite set of symbols corresponding to element names.*
- $\Sigma_m (\subseteq \Sigma_e)$ *is the set of symbols with omittable end tags.*
- M *is a function from* Σ_e *to regular languages over* Σ_e. *The regular language* $M(a)$ *corresponds to the content-model of* a.
- $r \in \Sigma_e$ *is the root element.*

We restrict the content-models for end-tag omittable elements: to the regular languages of the form $(a_1|a_2|\cdots|a_n)^*$ or $(a_1|a_2|\cdots|a_n)^+$ where a_i are symbols. This makes it possible to decide whether or not an end tag can be omitted by looking one symbol ahead. We say that an SGML DTD is *simple* if it satisfies this condition.

Let us consider the following simple DTD.

```
<!DOCTYPE a [
 <!ELEMENT a - - (x, b?)* >
 <!ELEMENT x - O (b)* >
 <!ELEMENT b - - (x)+ >
] >
```

This DTD is represented as $D = (\{a, x, b\}, \{x\}, M, a)$ where $M(a) = (xb?)^*$, $M(x) = b^*, M(b) = x^+$. The string $\acute{a}\acute{x}\grave{x}\acute{b}\acute{x}\grave{b}\grave{a}$ is valid with respect to D. On the

other hand, $\acute{a}\acute{x}\grave{x}\acute{b}\grave{b}\grave{a}$ is not valid, because this string does not have the x element that should appear inside the b element.

For simple DTDs, we formalize the language of a DTD D as follows.

1. Construct the regular hedge grammar $G_0(D)$ that generates all the valid balanced strings, in turn the valid strings without any tag omission.
2. Construct the finite transducer $T(D)$ that outputs all the valid strings obtained by omitting tags from a valid balanced string.

The language of D is the image of $G_0(D)$ under $T(D)$.

Regular hedge grammars (RHGs) were introduced by Murata as a model of XML schemas [Mur99]. Let us introduce a string version of RHGs.

Definition 2. *An RHG is a 5-tuple* $(\Sigma_2, \Sigma_1, N, P, S)$ *where* Σ_2*,* Σ_1*, and* N *are a base of a paired alphabet, the set of local symbols, and the set of nonterminals respectively. Each production rule in* P *has the following form:*

$$X \to \acute{a} R \grave{a}$$

where $X \in N$ *and* R *is a regular language over* $N \cup \Sigma_1$*.* $S \in N$ *is a start symbol* [1]*.*

An RHG defines a language over $\acute{\Sigma}_2 \cup \grave{\Sigma}_2 \cup \Sigma_1$. We denote elements of Σ_2 and Σ_1 by a, b, c and x, y, z, respectively. In this paper, without loss of generality we assume that each nonterminal of an RHG has exactly one production rule.

For the construction of the RHG $G_0(D)$, we consider that all element names are base symbols and introduce a nonterminal X_a for each element name $a \in \Sigma_e$. Then, $G_0(D)$ is defined as $(\Sigma_e, \emptyset, N, P, X_r)$ where $N = \{X_a \mid a \in \Sigma_e\}$ and P has the following rule for each $a \in \Sigma_e$.

$$X_a \to \acute{a} M(a) \grave{a}$$

This is basically the same as the interpretation used for XML DTDs by Berstel and Boasson [BB02].

Example 1. Let us consider the DTD D_1 below.

```
<!DOCTYPE a [
 <!ELEMENT a - - (x, b) >
 <!ELEMENT b - - (b)* >
 <!ELEMENT x - O (y | a)* >
 <!ELEMENT y - O (b)* >
] >
```

$G_0(D_1) = (\{a, b, x, y\}, \emptyset, \{X_a, X_b, X_x, X_y\}, P, X_a)$ where P has the following production rules.

$$X_a \to \acute{a} X_x X_b \grave{a} \quad X_b \to \acute{b} X_b^* \grave{b} \quad X_x \to \acute{x}(X_y | X_a)^* \grave{x} \quad X_y \to \acute{y} X_b^* \grave{y}$$

This RHG generates all valid balanced strings of the DTD.

[1] The original definition of RHGs allows us to use a regular expression instead of a single nonterminal to describe starting points of derivation.

To formalize end-tag omission, we introduce the finite transducer $T(D)$ that takes the valid balanced strings of the DTD D. The transducer produces all possible strings that can be obtained by omitting end tags according to D.

Let us first review a subclass of finite transducers called a generalized sequential machine (GSM). We adopt in this paper the definition of GSMs without final states.

Definition 3. *A GSM T is a 5-tuple $(Q, \Sigma, \Delta, \sigma, q_0)$ where Q is the finite set of states, Σ is the input alphabet, Δ is the output alphabet, σ is the transition-and-output function from $Q \times \Sigma$ to $2^{Q \times \Delta^*}$, and $q_0 \in Q$ is the initial state.*

For a simple DTD, the end tag of $a \in \Sigma_m$ can be omitted if the next symbol is an end tag $\grave{b}$ such that $b \neq a$, or a start tag $\acute{b}$ such that b does not appear in $M(a)$. Although the condition related to start tag differs from the original one in Section 2, they coincide for a simple DTD, because $M(a)$'s form is restricted to enable this.

We simplify presentation of $T(D)$ by describing it as taking a reversed string and outputting all the reversed valid strings obtained by end-tag omission. The transducer memorizes the last outputted symbol as its state and decides whether an end tag can be omitted or not. The reversed string of a valid string $\alpha_1\alpha_2\cdots\alpha_n$ is $\check{\alpha}_n\cdots\check{\alpha}_2\check{\alpha}_1$ where $\check{\acute{a}} = \grave{a}$ and $\check{\grave{a}} = \acute{a}$. The GSM $T(D)$ is formalized as follows.

Definition 4. *Let $D = (\Sigma_e, \Sigma_m, M, r)$ be a simple SGML DTD. We define the GSM $T(D)$ as $(Q, \Sigma_T, \Sigma_T, \sigma_T, q_0)$ where $\Sigma_T = \acute{\Sigma}_e \cup \grave{\Sigma}_e$ and $Q = \{q_0\} \cup \{q_\alpha \mid \alpha \in \Sigma_T\}$. The GSM has the following transitions and outputs.*

$(q_\alpha, \alpha) \in \sigma_T(q, \alpha)$ *if* $\alpha \in \Sigma_T$ *and* $q \in Q$.
$(q_0, \epsilon) \in \sigma_T(q_0, \acute{x})$ *if* $x \in \Sigma_m$.
$(q_{\acute{a}}, \epsilon) \in \sigma_T(q_{\acute{a}}, \acute{x})$ *if* $x \in \Sigma_m$, $a \in \Sigma_e$, *and* $a \neq x$.
$(q_{\grave{a}}, \epsilon) \in \sigma_T(q_{\grave{a}}, \acute{x})$ *if* $x \in \Sigma_m$, $a \in \Sigma_e$, *and* a *does not appear in* $M(x)$.

Then, The language of a simple DTD D is formalized as $(T(D)(L(G_0(D))^R))^R$.

It should be noted that the language cannot be represented by a regular hedge language in general. Let us consider the following DTD [2].

```
<!DOCTYPE a [
  <!ELEMENT a - O (a)? >
] >
```

The language of this DTD is $\{\acute{a}^m\grave{a}^n \mid 0 < m \wedge 0 \leq n \leq m\}$ and cannot be represented as an RHG.

4 A Translation from a DTD into an RHG

We consider a subclass of simple DTDs where the number of direct nestings of omittable elements is bounded and translate a DTD in this class into an RHG

[2] Although this DTD is not simple, the language can be formalized by slightly extending the definition of $T(D)$.

by considering tags of omittable elements as local symbols. This class includes the HTML DTD and thus enables validation of dynamically generated HTML documents with the decision algorithm of Minamide and Tozawa [MT06].

Let us consider the graph $\{(x, y) \in \Sigma_m \times \Sigma_m \mid y \text{ appears in } M(x)\}$ where Σ_m is the set of end-tag omittable elements. We say that a DTD is *acyclic* if its graph is acyclic. This requirement corresponds to requiring that the number of direct nestings of omittable elements is bounded. For an acyclic simple DTD, we develop a translation into an RHG. It consists of the three steps.

First, we construct the RHG $G'_0(D)$ that generates the same language as $G_0(D)$ by taking advantage of the acyclicity of DTDs. It is done by recursively expanding all nonterminals X_a for $a \in \Sigma_m$.

Example 2. For the DTD D_1, we obtain $G'_0(D_1) = (\Sigma'_e, \acute{\Sigma}_m \cup \grave{\Sigma}_m, \{X_c \mid c \in \Sigma'_e\}, P', X_a)$ where $\Sigma'_e = \Sigma_e \setminus \Sigma_m$ and P' has the following production rules.

$$X_a \to \acute{a}\acute{x}(\acute{y}X_b^*\grave{y}|X_a)^*\grave{x}X_b\grave{a} \quad X_b \to \acute{b}X_b^*\grave{b}$$

Hereafter in this section, we consider that the symbols in $\acute{\Sigma}_m \cup \grave{\Sigma}_m$ are local: $\Sigma_2 = \Sigma_e \setminus \Sigma_m$ and $\Sigma_1 = \acute{\Sigma}_m \cup \grave{\Sigma}_m$.

Second, we lift the GSM $T(D)$ over $\acute{\Sigma}_e \cup \grave{\Sigma}_e$ to $T_a(D)$ over $\{X_b \mid b \in \Sigma_e \setminus \Sigma_m\} \cup \acute{\Sigma}_m \cup \grave{\Sigma}_m$ for each $a \in \Sigma_e \setminus \Sigma_m$. Namely, we lift the GSM operating on the terminal symbols to $\Sigma_1 \cup N$. This lifting is possible if a GSM is *surface local*. A surface local GSM, as we define it, is a GSM that operate essentially on the surface of balanced string.

Definition 5. *Let $\Sigma = \acute{\Sigma}_2 \cup \grave{\Sigma}_2 \cup \Sigma_1$ and $T = (Q, \Sigma, \Sigma, \sigma, s)$ be a GSM. We say that T is surface local if the following conditions hold.*

1. *For each $a \in \Sigma_2$, there exist states $q_{\acute{a}}$ and $q_{\grave{a}}$ such that $\sigma(q, \acute{a}) = \{(q_{\acute{a}}, \acute{a})\}$ and $\sigma(q, \grave{a}) = \{(q_{\grave{a}}, \grave{a})\}$ for all $q \in Q$.*
2. *For all $b \in \Sigma_1$, $(q', w) \in \sigma(q, b)$ implies $w \in \Sigma_1^*$.*

It is clear from the definition that the GSM $T(D)$ for a simple SGML DTD D is surface local. A surface local GSM can be lifted as follows.

Definition 6. *Let Σ be $\acute{\Sigma}_2 \cup \grave{\Sigma}_2 \cup \Sigma_1$, $T = (Q, \Sigma, \Sigma, \sigma, s)$ be a surface local GSM, and $G = (\Sigma_2, \Sigma_1, N, P, S)$ be an RHG.*

We introduce a lifted GSM $T_a = (Q, \Sigma_1 \cup N, \Sigma_1 \cup N, \sigma', q_{\grave{a}})$ for each $a \in \Sigma_2$. The transition-output function σ' is defined as follows:

$$\sigma'(q, x) = \sigma(q, x) \quad (q \in Q \text{ and } x \in \Sigma_1)$$
$$\sigma'(q, X) = \{(q_{\acute{b}}, X)\} \ (q \in Q \text{ and } G \text{ has a rule of the form } X \to \acute{b}R\grave{b}).$$

Finally, we construct the RHG $G(D)$ that represents the language of D, by composing $T_a(D)$ and each production rule of $G'_0(D)$.

Theorem 1. *Let G and T be an RHG and a surface local GSM. We construct an RHG G' as follows:*

$$X \to \acute{a}T_a(R)\grave{a}$$

for a production rule $X \to \acute{a}R\grave{a}$. Then, $T(L(G)) = L(G')$.

In the previous section, we have defined the language of an SGML DTD as $(T(D)(L(G_0(D))^R))^R$. Let G' be an RHG constructed as in the theorem for $G'_0(D)^R$ and $T(D)$. Then we have $G' = T(D)(L(G_0(D))^R)$. Thus, the language of D can be described by the RHG G'^R.

Example 3. The grammar that generates the language of D_1 can be described as follows:

$$X_a \to \acute{a}\grave{x}(\acute{y}X_b^*\grave{y}?|X_a)^*\grave{x}X_b\grave{a} \mid \acute{a}\grave{x}((\acute{y}X_b^*\grave{y}?|X_a)^*(\acute{y}X_b^*\grave{y}|X_a))?X_b\grave{a}$$
$$X_b \to \acute{b}X_b^*\grave{b}$$

where X_a is the start symbol. The production rule for X_a can be read as follows. If $\grave{x}$ is omitted then the last $\grave{y}$ cannot be omitted, and vice versa.

5 Exclusion and Inclusion in SGML DTD

In contrast to XML DTD, SGML DTD can specify non-local constraints on elements with inclusion and exclusion. They are used to allow or disallow some elements appearing as a descendant of an element.

We have extended our translation from an SGML DTD to an RHG to support the exclusions appearing in the HTML DTD. To simplify translation, the support of exclusion is restricted so that exclusion is specified only for non-omittable elements. The HTML DTD satisfies this restriction.

Let us consider the following DTD to explain the extended translation.

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

The parts `-(b)` and `-(c)` are the specifications of exclusion. The former means `b` cannot appear as a descendant of `b`, even if it is specified as being allowed to do so. The latter indicates the same for `c` in `c`.

This DTD is translated into an RHG by introducing a nonterminal X_a^S for each element name a and the set of excluded elements S as follows:

$$X_a^{\emptyset} \to \acute{a}(X_b^{\emptyset}|X_c^{\emptyset})^*\grave{a} \quad X_a^{\{b\}} \to \acute{a}(X_c^{\{b\}})^*\grave{a} \quad X_a^{\{b,c\}} \to \acute{a}\grave{a}$$
$$X_b^{\emptyset} \to \acute{b}X_a^{\{b\}}X_c^{\{b\}}\grave{b} \quad X_b^{\{b\}} \to \acute{b}X_a^{\{b\}}X_c^{\{b\}}\grave{b}$$
$$X_c^{\emptyset} \to \acute{c}X_a^{\{c\}}\grave{c} \qquad X_c^{\{b\}} \to \acute{c}X_a^{\{b,c\}}\grave{c}$$

where $X_b^{\{b,c\}}$ is not included because it generates no terminal strings.

The transducer $T(D)$ for the DTD D can be constructed in the exactly same manner under the condition that exclusion is specified only for non-omittable elements.

One problem with this translation is that it may increase the size of the grammar exponentially. In our experiments, we could obtain an RHG for the HTML DTD even if we enabled exclusion. However, we failed to minimize the RHG and to use it directly for HTML validation.

6 Experimental Results

We have implemented a parser for SGML DTD and the translation from DTDs into RHGs given above. We conducted our experiments on the HTML 4.01 Transitional DTD, which contains declarations for 109 elements. To simulate unsupported features of DTDs appearing in the HTML DTD, we modified the DTD as follows.

- The TABLE element is defined as follows. The start and end tags of TBODY can be omitted and they usually are omitted.

```
<!ELEMENT TABLE - -
     (CAPTION?, (COL*|COLGROUP*), THEAD?, TFOOT?, (TBODY)+)>
<!ELEMENT TBODY    O O (TR)+           -- table body -->
```

  To support the omission, we have replaced the `(TBODY)+` part of the content-model of the TABLE with `(TBODY|TR)+` and conducted our experiments.
- The inclusion feature of the SGML DTD is used in the definition of the header of HTML documents. We have simulated it by expanding the element declarations with included elements. Although the inclusion feature is also used for the INS and DEL elements, we ignored them because the programs we considered do not use the INS and DEL elements.

In the implementation, we represent an RHG as a grammar with production rules of the forms $X \to \acute{a}Y\grave{a}Z$ or $X \to bY$ or $X \to \epsilon$, where $a \in \Sigma_2$ and $b \in \Sigma_1$. A grammar in this form can be considered as a tree automaton and algorithms such as determinization and minimization for tree automata can be applied to it. An RHG can be converted into a grammar in this form, and vice versa. In our implementation, a minimized RHG is converted into an algebra called a binoid to decide CFL-RHL inclusion [PQ68].

When we ignored exclusions we obtained the RHG that has 187 nonterminals and 7471 production rules. After minimizing it, the grammar has 56 nonterminals and 6758 production rules and the binoid converted from the RHG has 2381 elements. The RHG obtained with exclusion enabled translation has 3213 nonterminals and 115647 production rules. As we noted earlier, the translation to support exclusion may increase the size of the RHG exponentially. Although the RHG is only one order of magnitude larger, the determinization and minimization of the RHG and the generation of the binoid failed because the RHG is too large.

The implementation was incorporated into 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 HTML DTD. To reduce the size of the binoid obtained by the translation, 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.

Table 1. HTML validation

Programs	Element names	RHG (minimized) Nonterminals	Productions	Binoid	Bugs	Execution time (s)
webchess	20	20	439	375	1	3.37
faqforge	19	16	315	106	16	1.19
phpwims	18	16	268	39	3	1.65

Table 1 shows the results of our experiments. The column 'elements' shows the number of element names that may appear in generated HTML documents. The columns 'RHG' and 'binoid' show their sizes. We found several bugs through our experiments and corrected them. The numbers of bugs found are also shown. The column 'time' shows the execution time spent to generate binoids from the DTD and check the CFL-RHL inclusion. These times do not include the time spent to generate a CFG from a PHP program by the analyzer.

References

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

[Gol90] Goldfarb, C.F.: The SGML Handbook. Oxford University Press, Oxford (1990)

[Min05] Minamide, Y.: Static approximation of dynamically generated Web pages. In: Proceedings of the 14th International World Wide Web Conference, pp. 432–441. ACM Press, New York (2005)

[MT06] Minamide, Y., Tozawa, A.: XML validation for context-free grammars. In: Kobayashi, N. (ed.) APLAS 2006. LNCS, vol. 4279, pp. 357–373. Springer, Heidelberg (2006)

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

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

[Wor99] World Wide Web Consortium. HTML 4.01 Transitional DTD (1999), `http://www.w3.org/TR/html401/loose.dtd`

[WvE89] Warmer, J., van Egmond, S.: The implementation of the Amsterdam SGML Parser. Electronic Publishing 2(2), 65–90 (1989)