動的に生成される文書のXHTML妥当性検査

Validating Dynamically Generated XHTML Documents

乙井 信男 † 南出 靖彦 †

Nobuo OTOI, Yasuhiko MINAMIDE

† 筑波大学システム情報工学研究科

Graduate School of Systems and Information Engineering, University of Tsukuba

{otoi,minamide}@score.cs.tsukuba.ac.jp

これまで HTML の妥当性検査は静的な Web ページに対し適用され、動的に Web ページが生成される場合検査する手法はなかった．そこで本論文ではサーバーサイドプログラムの出力をプログラム解析の手法を用いて文脈自由文法に近似し、動的な Web ページが XHTML の仕様を満たすか検査する手法を提案、実装する．仕様を表すオートマトンを単純に構成すると状態数が指数関数的に増えてしまい、実用に耐えられないことから、DTD の局所性を用いた検査を実装する．

1 はじめに

動的な Web ページは静的な Web ページよりもクライアントの要求に柔軟に対応できることから、多くの Web サイトで用いられている．動的な Web ページはサーバーサイドで生成され、このとき用いられるプログラムはサーバーサイドプログラムと呼ばれる．動的な Web ページは柔軟な表現ができる一方で、適切な処理をしないと Web ページに脆弱性などの問題が含まれる可能性がある．

こうした背景から、サーバーサイドプログラムを解析し、その性質を検証する研究が行われている [2]．我々が代表的なサーバーサイド言語である PHP を対象にしたプログラムの文字列解析器を開発している [4] ．この解析器は、PHP プログラムが出力する可能性のある文字列の集合を文脈自由文法を用いて近似する．この解析によって得られた文脈自由文法をもとに、動的に生成される Web ページを検査することができる．例えば、クライアントのどのような要求に対しても PHP プログラムが必ず仕様を満たす文字列を出力するかチェックすることで安全性を検査することができる．

一方、サーバーサイドプログラムの抱える問題として、妥当性の問題がある．静的な Web ページが HTML や XHTML の規格に対する妥当性を検査するツールは数多く利用されているが、動的に生成する文書では、Web ページを静的に決定できないために妥当性の検査をすることができなかった．本研究では我々が開発している文字列解析器を用いることで PHP プログラムを解析し、そのプログラムが生成する文書の XHTML 妥当性を検査する．

今回提案する検査では、文字列解析器の出力する文脈自由言語が XHTML の妥当性を満たすか検査する．XHTML に適合する文字列の集合は文脈自由文法を用いて表す事ができるので、文脈自由言語同士の包含関係が決定できれば妥当性を検査できる．しかしこの包含関係は決定不能問題であり、単純に検査できない．そこで本研究では対象とする文章の深さを制限する事で検査を可能にする．

2 文字列解析器

近年、プログラムの生成する文字列の集合を近似する研究が行われている [1][5]．我々もまた、PHP プログラムを対象に、文字列の集合を文脈自由言語に近似して出力する文字列解析器を開発している．この解析器には PHP プログラムと入力に関する仕様を与える必要がある．解析器の動作を示すために、以下に PHP プログラムの例を示す．

```
<?php
for ($i = 0; $i < $n; $i++)
  $x = "<table><tr><th>".$x.
         "</th></tr></table>";
echo $x;
?>
```

PHP のプログラムは<?で始まり、?>で終わる．変数名は$から始め、この例では$x, $i, $n が変数である．

このプログラムに文字列解析器を適用するためには、変数の初期値を仕様として与える必要がある．仕

様には以下のように正則表現か変数の型を書く事ができる．

```
$x:  /abc|xyz/
$n:  int
```

ここでは、$x の値は abc または xyz、$n は int 型であることを表している．解析器は、これらの情報を基にプログラムの出力を文脈自由文法を用いて近似する．この例では、以下のような生成規則が得られる．

$$
\begin{aligned}
X &\rightarrow abc \\
X &\rightarrow xyz \\
X &\rightarrow \langle table\rangle\langle tr\rangle\langle th\rangle X\langle /th\rangle\langle /tr\rangle\langle /table\rangle
\end{aligned}
$$

このプログラムは、テーブルタグが n 回ネストする文字列を出力する．一方、上の文脈自由法は、下の文字列の集合と等価であり、正しい近似になっている．

$$
\begin{aligned}
&\{(\langle table\rangle\langle tr\rangle\langle th\rangle)^n abc(\langle /table\rangle\langle /tr\rangle\langle /th\rangle)^n \mid n \geq 0\}\cup \\
&\{(\langle table\rangle\langle tr\rangle\langle th\rangle)^n xyz(\langle /table\rangle\langle /tr\rangle\langle /th\rangle)^n \mid n \geq 0\}
\end{aligned}
$$

上のように得られた文脈自由文法を用いてプログラムの性質を検査することができる．

3 XHTML と DTD

本研究で妥当性を検査する XHTML とは、HTML を XML の文法に適合するよう定義し直したマークアップ言語である．HTML の文法は複雑であるのに対し、XHTML の文法はシンプルなので妥当性の検査が簡単になる．XHTML の文法は、XML 文書の構造を定義する DTD(Document Type Definition) を用いて与えられている．以下に、XHTML の DTD の一部を単純化したものを示す．DTD では、

```
<!ELEMENT table (tr)>
<!ELEMENT tr (th|td)>
<!ELEMENT th (table)?>
<!ELEMENT td (table)?>
```

図 2: DTD の例

ELEMENT で要素宣言を行う．図 2 の 1 行目では要素 table が内容モデル tr をとっている．これは、⟨table⟩⟨tr⟩...⟨/tr⟩⟨/table⟩ のように、タグ table の中にタグ tr が現れることを表す．同様に 2 行目では要素 tr の内容モデルが (th|td) となっており、これはタグ th, td のどちらか 1 つをを内部に持たなくてはならないことを意味する．この DTD に対して妥当性を満たす最小の文書は ⟨table⟩⟨tr⟩⟨th⟩⟨/th⟩⟨/tr⟩⟨/table⟩ と ⟨table⟩⟨tr⟩⟨td⟩⟨/td⟩⟨/tr⟩⟨/table⟩ である．

4 XHTML 妥当性検査

本研究は、PHP プログラムの出力する Web ページが必ず XHTML の仕様を満たすか検査する．この検査は PHP プログラムの出力する文字列を表す文脈自由文法 G に対し、$L(G) \subset L(XHTML)$ で表すことができるが、この関係を直接調べることはできない．ここでの問題は、XHTML の仕様は本質的に文脈自由文法であり、文脈自由文法同士の包含関係は決定不能問題であることである．

この問題を解決するために、PHP プログラムによって生成される XHTML 文書のタグのネストの深さが有限の場合に関して、検査を行うことにする．多くのサーバーサイドプログラムは、テーブル構造を持つ Web ページを生成するので、生成する Web ページの深さは、有限になると考えている．

深さを制限することで、妥当な XHTML 文書の集合は正則言語で表すことができる．深さ i 以下の妥当な XHTML 文書の集合を $XHTML_i$ とすると以下の検査を行うことになる．

$$L(G) \subset L(XHTML)_i$$

この包含関係の検査は、右辺が正則言語の場合は検査可能であり、本研究では、Melski と Reps によるアルゴリズム [3] を用いた．

具体的には、深さが i 以下の XHTML 文書を表すオートマトン FA_i を、XHTML の DTD から生成した．オートマトン FA_i は、図 1 のように、まず、各タグと深さにに対して、対応する正則言語を求めることによって構成した．

しかし、XHTML の DTD に対して、直接に上のオートマトンを構成すると、その DTD が???大きいのでの、巨大なオートマトンができる．一方、解析対象の PHP プログラムの文法 L(G) が HTML 上の全てのタグを出力する文法とは限らない．そこで、DTD の文法から検査に不必要なタグに関する文法を取り除いた DTD' を構成し、DTD' が表す言語 XHTML' に対して $L(G) \subset XHTML'_i$ を検査する．

また、実際に検査を行う場合には、前もって、生成される文書の深さの最大値はわからない．そこで、妥当性の検査は検査する最大の深さを定め、深さ 1 から順に検査を行った．

$\text{table}_0 \to \phi$				$\text{table}_2 \to \langle\text{table}\rangle\text{tr}_1\langle/\text{table}\rangle$	=	ϕ
$\text{tr}_0 \to \phi$				$\text{tr}_2 \to \langle\text{tr}\rangle(\text{th}_1+\text{td}_1)\langle/\text{tr}\rangle$	=	$\langle\text{tr}\rangle(\langle\text{th}\rangle\langle/\text{th}\rangle + \langle\text{td}\rangle\langle/\text{td}\rangle)\langle/\text{tr}\rangle$
$\text{th}_0 \to \phi$				$\text{th}_2 \to \langle\text{th}\rangle(\epsilon+\text{table}_1)\langle/\text{th}\rangle$	=	$\langle\text{th}\rangle\langle/\text{th}\rangle$
$\text{td}_0 \to \phi$				$\text{td}_2 \to \langle\text{td}\rangle(\epsilon+\text{table}_1)\langle/\text{td}\rangle$	=	$\langle\text{td}\rangle\langle/\text{td}\rangle$
$\text{table}_1 \to \langle\text{table}\rangle\text{tr}_0\langle/\text{table}\rangle$	=	ϕ		$\text{table}_3 \to \langle\text{table}\rangle\text{tr}_2\langle/\text{table}\rangle$	=	$\langle\text{table}\rangle\langle\text{tr}\rangle(\langle\text{th}\rangle\langle/\text{th}\rangle$
$\text{tr}_1 \to \langle\text{tr}\rangle(\text{th}_0+\text{td}_0)\langle/\text{tr}\rangle$	=	ϕ				$+\langle\text{td}\rangle\langle/\text{td}\rangle)\langle/\text{tr}\rangle\langle/\text{table}\rangle$
$\text{th}_1 \to \langle\text{th}\rangle(\epsilon+\text{table}_0)\langle/\text{th}\rangle$	=	$\langle\text{th}\rangle\langle/\text{th}\rangle$		$\text{tr}_3 \to \langle\text{tr}\rangle(\text{th}_2+\text{td}_2)\langle/\text{tr}\rangle$	=	$\langle\text{tr}\rangle(\langle\text{th}\rangle\langle/\text{th}\rangle + \langle\text{td}\rangle\langle/\text{td}\rangle)\langle/\text{tr}\rangle$
$\text{td}_1 \to \langle\text{td}\rangle(\epsilon+\text{table}_0)\langle/\text{td}\rangle$	=	$\langle\text{td}\rangle\langle/\text{td}\rangle$		$\text{th}_3 \to \langle\text{th}\rangle(\epsilon+\text{table}_2)\langle/\text{th}\rangle$	=	$\langle\text{th}\rangle\langle/\text{th}\rangle$
				$\text{td}_3 \to \langle\text{td}\rangle(\epsilon+\text{table}_2)\langle/\text{td}\rangle$	=	$\langle\text{th}\rangle\langle/\text{th}\rangle$

図 1: タグから正則言語へのテーブル

この検査を実装し、SourceForge から配布されているオープンソースプログラム WebCalendar に対して実験を行った. 実験はこのプログラムのメインページを生成する month.php を対象にし、DTD には DTD XHTML 1.0 Transitional を用いた. 予備的な実験により、このプログラムは深さ 7 の文書を生成することがわかっている. 結果はタグの深さ 7 で検査中、マシンのメモリーが不足して検査は失敗した. 以下は、この実験で得られたタグの深さとオートマトンの状態数の関係である.

タグの深さ	3	4	5	6	7
状態数	276	1475	8283	36369	143237

ここから、オートマトンの状態数がタグの深さに対し指数関数的に増えていることがわかる. 一方、検査で用いている文脈自由言語と正則言語の共通部分の計算がオートマトンの状態数の 3 乗のアルゴリズムであることから、タグのネストが深くなることで計算量が膨大になったと考えられる.

また、この実験の中で、DTD の 1 回以上繰り返しを表す表現 X^+ に対応するプログラムに関して、文字列解析器が不十分な精度の文脈自由文法を生成することが多いことがわかった. 厳密な妥当性の検査ではなくなるが、本研究では、DTD の 1 回以上繰り返しを表す表現 X^+ を X^* として検査を行った.

5 DTD の局所性を用いた検査の改良

前節の実験によってオートマトンの状態数がタグの深さに対し指数関数的に増えてしまうことが分かった. そこで状態数の増加を DTD の局所性を用いて抑制する手法を考える. 状態数が爆発した原因は、DTD によって表される制約を一つのオートマトンで表すと、タグのネストの深さに対し指数関数的にその状態数が増加してしまうためだと考えられる. 一方、DTD には、ある要素とその子要素の関係しか記述できないという局所性がある. すなわち、あるタグの孫として、ある別のタグが必要であるというような制約は記述できない.

そこで、各深さ k ごとに、深さ k と $k+1$ について、親子関係の制約を検査する. 具体的には、$XHTML_i$ に対して、上の制約のみを検査するオートマトン $FA'_{i,k}(1 \le k < i)$ を作成し、各 k について包含関係を確かめる. これにより、検査時の状態数を減らしながら、妥当性の検査を行うことができる.

$L(G) \subset FA_i \Leftrightarrow$

$L(G) \subset FA'_{i,1} \wedge \cdots \wedge L(G) \subset FA'_{i,i-1}$

オートマトン $FA'_{i,k}$ は $R(x), f(DTD)$ および $g(C)$ を用いて以下のように定義する. ただし e_n, C_n はそれぞれ DTD における要素と内容モデルを表し、<!ELEMENT e_n C_n>を意味する.

$$R(x) = (\underbrace{\langle.*\rangle...(\langle.*\rangle}_{x\text{ 個}}\langle/.*\rangle)^*...\langle/.*\rangle)^*$$

$$f(DTD) = \langle e_1\rangle g(C_1)\langle/e_1\rangle + \langle e_2\rangle g(C_2)\langle/e_2\rangle + \cdots$$

$$g(C) = \begin{cases} \langle a\rangle R(i-k-1)\langle/a\rangle & C = a \\ g(D) + g(E) & C = D + E \\ g(D) + \epsilon & C = D? \\ g(D)g(E) & C = DE \\ (g(D))^* & C = D^* \\ (g(D))^* & C = D^+ \end{cases}$$

$$FA'_{i,k} = (\underbrace{\langle.*\rangle...(\langle.*\rangle}_{k-1\text{ 個}} f(DTD)\langle/.*\rangle)^*...\langle/.*\rangle)^*$$

このアルゴリズムで得られる正則言語は、ある深さ k と $k+1$ 以外のタグは常に左右のバランスしかチェックしないため、深さに対して線形の状態数をもつオートマトンで表すことができる. 例えば、このアルゴリズムを図 2 の DTD に対して適用し、$FA'_{3,1}$

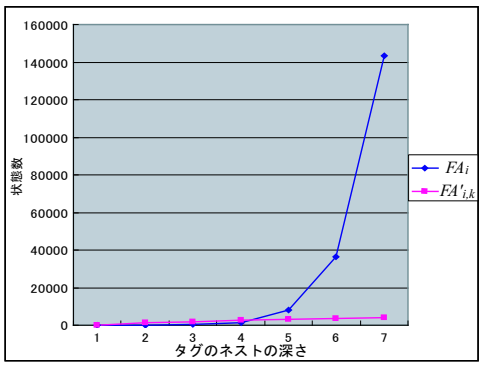

図 3: WebCalendar 検査時の状態数比較

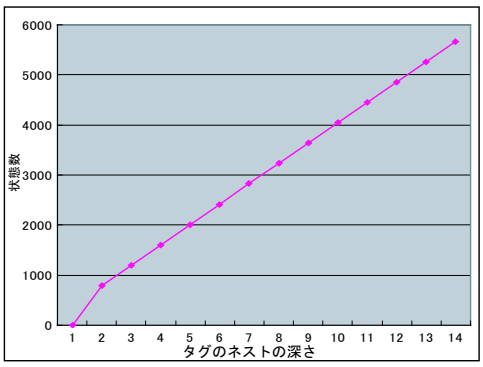

図 4: phpScheduleIt 検査時の状態数

を求めると、以下の正則表現が得られる.

$$\langle table\rangle\langle tr\rangle(\langle .*\rangle\langle /.*\rangle)^*\langle /tr\rangle\langle /table\rangle$$
$$+\langle tr\rangle(\langle th\rangle(\langle .*\rangle\langle /.*\rangle)^*\langle /th\rangle + \langle td\rangle(\langle .*\rangle\langle /.*\rangle)^*\langle /td\rangle)\langle /tr\rangle$$
$$+\langle th\rangle(\langle table\rangle(\langle .*\rangle\langle /.*\rangle)^*\langle /table\rangle + \epsilon)\langle /th\rangle$$
$$+\langle td\rangle(\langle table\rangle(\langle .*\rangle\langle /.*\rangle)^*\langle /table\rangle + \epsilon)\langle /td\rangle$$

6 実験

6.1 状態数の変化

前節の検査アルゴリズムを用いて、SourceForge から配布されているオープンソースプログラム WebCalendar と phpScheduleIt に対して実験を行った.

WebCalendar に関して、FA_i の状態数と、$FA'_{i,k}$ の状態数をグラフで表したものが図 3 である. ただし $FA'_{i,k}$ は同じ深さのオートマトンの中で最も状態数が大きくなる値 k を選んだ場合である. FA_i では深さとともに指数関数的に状態数が増えるのに対し、$FA'_{i,k}$ ではほぼ線形にとどめることができた.

この実験の中で、文字列解析器の近似が粗すぎるために検査が失敗する場合、また、WebCalendar に対応しない開始タグと終了タグを出力するバグがあるため検査が失敗する場合があった. 上の実験は、より正確に近似できるプログラムに書き換え、また発見したバグは修正し行った. これらに関しては、次節で述べる.

さらに、よりタグのネストが深い例として、PHP で書かれたスケジュール管理プログラム phpScheduleIt の検査を行った. 前準備として、関数 eval などのプログラム解析をするうえでサポートするのに困難な関数を同等なプログラムに置き換えた. この検査結果を図 4 に示す. タグのネストがより深いこの例においても状態数の増加は線形であることが確認できた. さらに検査によりこのプログラムの生成する XHTML 文書は妥当性を満たすことがわかった.

6.2 解析精度とバグの検出

前節で述べた実験において、文字列解析を行い、十分な解析精度で妥当性の検査を行うためには、プログラムを書き換える必要がある箇所がいくつかあった. ここでは、WebCalendar の検査について述べる. まず、以下のように if 文を用いてある条件が成り立つときに

```
if ( $pri == 3 ) echo "<strong>";
...
if ( $pri == 3 ) echo "</strong>\n";
```

対応する開始タグと終了タグを出力するものである. 文字列解析器は、if 文の条件部分を無視するため、妥当性の検査に失敗する. そこで、このプログラムは、以下のように書き換え検査を行った.

```
if ( $pri == 3 ) {
  echo "<strong>";
  foo(...)
  echo "</strong>\n";
} else
  foo(...);
```

ここで、関数 foo は、上のプログラムの ... 部分を実行する関数として定義した.

また、配列の操作に関して十分な精度が得られない部分も 2 箇所見つった. それらも、同じ動作を行うがより正確に近似できるプログラムに書き換え実験を行った.

これらの書き換えを行っても、WebCalendar の検査は失敗した. そのため、検査によって得られた反例

とプログラムを比較し、プログラムに対応しない開始タグと終了タグを出力するバグがないか調べ、そのようなバグが 2 箇所あることがわかった．このバグは、WebCalendar の開発者に報告した．

7 まとめ

PHP の文字列解析器によって得られる文脈自由文法を用いて XHTML の妥当性を検査する手法を提案し、実装した．さらに、オープンソースの PHP プログラムに対して実験を行った結果、DTD から生成されるオートマトンの状態数が指数関数的に増えることが問題となった．それに対してオートマトンをタグの深さを 2 ステップずつ検査するように分割することで状態数の爆発を防ぐ改良手法を提案した．これによってタグのネストが深いプログラムに対しても検査することができるようになり、妥当性を満たさないプログラムを見つけることができた．

提案した検査手法は、いくつかの意味で厳密でない．4 節で述べたように文字列解析器が言語中の 1 回以上繰り返しを 0 回以上繰り返しとして近似してしまうため、DTD 中の内容モデルも合わせて X^+ を X^* とみなして検査を行った．この点で妥当性の検査精度が低くなってしまった．今後は X^+ を X^+ として検査するには、どれくらい PHP プログラムを書き換える必要があるか調べたい．また、DTD に記述される情報は要素の親子関係だけではない．要素の属性に関する検査を行いたい．

参考文献

[1] Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach. Precise analysis of string expressions. In *Proc. 10th International Static Analysis Symposium, SAS '03*, Vol. 2694 of *LNCS*, pp. 1–18. Springer-Verlag, June 2003.

[2] Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, Der-Tsai Lee, and Sy-Yen Kuo. Securing web application code by static analysis and runtime protection. In *Proceedings of the 13th international conference on World Wide Web*, pp. 40–52, USA, 2004.

[3] David Melski and Thomas Reps. Interconvertibility of a class of set constraints and context-free language reachability. *Theoretical Computer Science*, Vol. 248, No. 1–2, pp. 29–98, 2000.

[4] Yasuhiko Minamide. Static approximation of dynamically generated Web pages. In *Proc. 14th International World Wide Web Conference*, pp. 432–441, May 2005.

[5] N. Tabuchi, E. Sumii, and A. Yonezawa. Regular expression types for strings in a text processing language. In *Proceedings of Workshop on Types in Programming (TIP)*, pp. 1–18, 2002.