インラインスクリプトを含んだXHTML文書に対するデータフロー解析を用いた構文検証手法

松下 誠[†] 鷲尾和則[†,☆] 井上克郎[†]

HTMLやXHTML文書の多くは，JavaScriptやPHPといったスクリプトを用いて動的に文書の内容を生成することが多い．これらの文書に対して，従来の構文検証手法はスクリプトの内容は無視されていたため実際には構文誤りを含む文書を含まない文書と誤って判断していた．そこで本研究では，インラインスクリプトとしてECMAScriptを含むXHTML文書の構文を検証するために，出力文のデータフロー解析を用いて出力文字列を正規表現を用いて表記し，それに対して検証するという手法を提案する．また，本手法を用いた構文検証ツールの実装を行い，実際の文書をツールに適用して手法の評価を行った．その結果，既存の検証ツールでは検出できなかった誤りを検出できることを確認した．

Data-flow Based Syntax Validation Method for XHTML Documents with Embedded Inline Scripts

Makoto Matsushita,[†] Kazunori Washio[†,☆] and Katsuro Inoue[†]

Today many HTML and XHTML documents are often dynamically generated by scripts such as JavaScript and PHP. The existing verification techniques cannot be applicable to these scripts. In order to verify the syntax of an XHTML document containing an in-line script, we propose a technique that we use the data flow analysis of the output sentences, create pattern of the output strings, and verify the pattern. In addition, we have been implemented the syntax verification tool using this technique, and we have evaluated our approach. As a result, we could find errors which have not been detected by the existing syntax analysis tools.

1. はじめに

近年，情報の可搬性や構文の単純さなどの観点から，eXtensible Hyper Text Markup Language (XHTML)[1)] を用いた文書が多く作成されている．一般的に，XHTML文書は「静的コンテンツ」と「動的コンテンツ」の2種類の内容で構成されている．ここで静的コンテンツとは，XHTMLで定められた構文のみで記述された内容が変更されないものであり，動的コンテンツとは，文書を閲覧する環境に依存して変化する内容であり，XHTMLとは独立に定義された言語によって記述されるプログラム（インラインスクリプト）によって記述されているものとする．

XHTML文書の内容はXHTMLのタグ付け規則（Document Type Definition: DTD）によって定義されている．このため，DTDに違反したタグ付けが行われた場合には，XHTMLブラウザによって正しく表示ができないなど，文書を利用する場合に大きな支障をきたす．このため，XHTML文書がDTDの定義に反しないように記述されているかどうかを検証するのは非常に重要である．

動的コンテンツの多くは，単純なテキスト情報だけではなく，XHTMLで定められたタグをも含む内容である．このため，もしインラインスクリプトを誤って記述した場合，一見XHTML構文誤りを含まない文書であっても，動的コンテンツの出力する内容によって文書全体が構文違反となる可能性がある．しかし，既存のXHTML構文検証手法は，XHTML文書の内容のうち静的コンテンツのみを対象とした検証であり，動的コンテンツは単なるテキストデータと見なされる．このため，本来構文誤りを含むXHTML文書を構文誤りのない文書と誤認することがあるため，検証結果

† 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University
☆ 現在，三菱電機先端技術総合研究所
Presently with Advanced Technology R&D Center, Mitsubishi Electric Corporation

の信頼性が低くなっていた．

そこで本研究では，動的コンテンツとして ECMAScript で記述されたインラインスクリプトを含んだ XHTML 文書に対して，動的コンテンツの内容を考慮した構文検証手法の提案を行う[2]．本手法ではまず，対象となる XHTML 文書を静的コンテンツと動的コンテンツを分割する．次に，動的コンテンツとして記述されているインラインスクリプトに含まれている，文字列の出力を行う文（以降，出力文）中の変数に対し，データフロー解析と構文解析を行う．両方の解析結果を組み合わせ，動的コンテンツが出力する内容を正規表現を用いて記述する．さらに，静的コンテンツを解析することによって得られるタグ情報と，動的コンテンツから得られたタグ情報を結合した結果から，文書全体のツリーを生成し，このツリーに対して DTD に照らした構文検証を行う．また，本手法を用いた XHTML 構文検証ツール ECMAX の実装を行い，実際に利用されている XHTML 文書の検証実験を通じて，本手法の有効性評価を行う．

2. 構造化文書とインラインスクリプト

本章では，構造化文書を記述するマークアップ言語およびインラインスクリプトについて説明する．また，インラインスクリプトを含んだ構造化文書の構文検証についても触れる．

2.1 XHTML

Hyper Text Markup Language（HTML）[3] はウェブページ作成のために，W3C が標準化したマークアップ言語であるが，タグの省略が可能など厳密にその構文が定義されていないため構文の曖昧さが存在した．そこで W3C が既存の HTML と仕様が等価な言語を XML を用いて定めた言語が XHTML である．XHTML はタグの省略が不可能であるなど，厳密な言語定義がなされており，近年のウェブページ作成時における標準言語となっている．

2.2 ECMAScript

ECMAScript [4] は，JavaScript をもとに European Computer Manufacturer Association（ECMA）によって標準化されたインラインスクリプト用言語である．ブラウザに固有のメソッドなどを排除したものとなっている．

2.3 検証の難しさ

一般的に，構造を持つ文書は，定められた構文に従って正しく記述されていることが重要である．その理由として，誤って構造が記述された文書は，受け手が文書の意味を誤解したり，内容が読み取れないなどといった問題が発生することがあげられる．このことから，その構文を検証するシステムが数多く存在している．しかし，既存の検証システムは，インラインスクリプトの内容は単なる文字列データと見なし，その内容まで検証しない．したがって，インラインスクリプトが誤った構文の文書を生成しても，これらの検証システムは構文誤りを判断ができない．このため，インラインスクリプトの内容も検証するシステムが必要となる．

我々が以前提案したように[5]，入力として適当なデータを与えてインタプリタを介すれば，インラインスクリプトが生成する文書も含めて検証することが可能である．しかし，テストデータによって実行結果が変わるため，検証結果も変わる可能性がある．よってすべての実行経路を通るテストデータによって検証を行う必要があるが，そのようなテストデータを作成することは一般的に困難である．

3. 関連研究

動的に生成される HTML 文書や XML 文書について，その構文を検証する研究がいくつか行われている．

Brabrand らは <bigwig> [6] という HTML 文書や XML 文書を生成するためのスクリプト記述言語を対象として，スクリプトが動的に生成する HTML（XHTML）文書の構文検証を行った．

Kempa らは DOM [7] を拡張し，DOM 自身に構文の正しさを保証する仕組みを持たせた V-DOM と，V-DOM 用の XML 処理言語である P-XML [8] の提案を行った．

Mejier らは XML 処理用の関数型言語 XM λ [9] を開発した．また，Hosoya らは XML 処理のための型付き関数型言語 XDuce [10] を開発し，型チェック機能を導入しプログラムが生成する XML 文書について構文の正しさを保証するようにした．

いずれの研究も，XML 文書や HTML 文書を生成するために新しい言語を作り，その言語に構文の正しさを保証するような仕組みを構築している．しかし，プログラム言語として独自の言語を対象としており，さまざまな XHTML 文書で利用可能である実用的な手法とはいえない．

4. 提案する検証手法

提案する検証手法は，XHTML の構文を定義した DTD と，XHTML 文書を入力として，文書を DTD に照らして静的な検証を行う．検証の結果，構文違反が文書中に存在したかどうか，また違反 ga 存在する

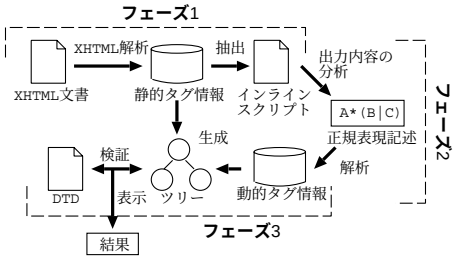

図 1 検証手法の流れ
Fig. 1 Work flow of validation technique.

場合にはその場所についての情報を出力として得る．

具体的には，XHTML 文書中に含まれている，その文字列出力が正規表現として表される ECMAScript インラインスクリプトを対象に検証を行う．一般的に，プログラムが出力する文字列を正規表現で表すことはできないが，本手法の目的が XHTML の文書と付随するプログラムの出力をあわせて，正規表現として表現される DTD に照らした検証を行うことであるため，正規表現で表現できる範囲のみを対象としても問題とはならないと考えている．本手法は動的に変わりうる出力文字列を正規表現を用いて表すため，可能性のあるすべての実行経路を検証できる．ただし，動的に変数の値が決定され，それが出力に影響を与える場合，静的な検証は行えない可能性がある．この場合，本手法はその部分についてプログラムの出力が定まらないものとして扱う．

本手法の概略を**図 1** に示す．この手法は大きく分けて次の 3 段階からなる．

フェーズ 1：静的コンテンツ解析 XHTML 文書の静的コンテンツについて構文解析を行い，タグ情報とインラインスクリプトを抽出する．

フェーズ 2：動的コンテンツ解析 ECMAScript で記述されているインラインスクリプトの解析を行う．これは構文解析および出力文に含まれる変数のデータフロー解析が含まれる．それらの結果をもとに，繰返し（*）および選択（|）記号を用いた正規表現として出力文字列を表現する．

フェーズ 3：構文検証 最後に，フェーズ 2 で得られた正規表現を解析し，タグ情報を生成する．その後フェーズ 1 で得られたタグ情報とあわせて，文書を要素の親子関係として表したツリーを生成し，DTD の要素定義に基づいて構文の検証を行う．

以下，3 つのフェーズについて順に説明する．

4.1 フェーズ 1：静的コンテンツ解析

本フェーズでは XHTML 文書の構文解析を行う．イベント駆動型の XML パーサを用いて構文解析を行い，

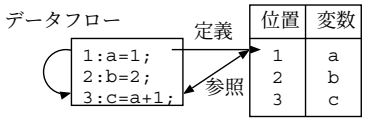

図 2 データフロー解析
Fig. 2 Data flow analysis.

XHTML タグの要素名や属性リスト，テキスト要素に含まれる文字列の情報（以降，タグ情報と呼ぶ）と，それらの XHTML 文書上での位置情報を取得する．

4.2 フェーズ 2：動的コンテンツ解析

本フェーズでは，以下の手順でインラインスクリプトの解析を行う．まず，インラインスクリプトの構文解析および意味解析を行う．次に，出力文に含まれる変数に対してデータフロー解析を行う．最後にデータフロー解析から得た変数の値をもとに，出力される文字列を，構文解析で得られた制御構造により正規表現を用いて表す．

4.2.1 構文解析および意味解析

ECMAScript の言語仕様[4] をもとにインラインスクリプトの構文解析を行い抽象構文木を作成する．また，同時にインラインスクリプトの意味解析を行う．このとき，抽象構文木の各ノードに対して，インラインスクリプト上での位置情報も記録する．

4.2.2 データフロー解析

データフロー解析では，インラインスクリプトに含まれる変数の代入および参照関係の解析を行う．たとえば**図 2** の左側に示されるインラインスクリプトの場合，まず，1 行目を解析して，位置 1 で変数 a が定義されていることを変数表（図 2 右側）に記録する．2 行目も同様に解析し，位置 2 で変数 b が定義されていることを記録する．次に 3 行目の解析が行われると，代入文の左辺で参照される変数 a の定義を変数表から探す．その結果，1 行目であることが分かるため，1 行目から 3 行目に対してデータフローが存在することが分かる．これを順にプログラムの先頭から繰り返すことによって，依存グラフ[11] を作成する．

4.2.3 正規表現を用いた出力文字列の表記

正規表現による表記とは，インラインスクリプトの出力文字列を，正規表現のメタ文字を含む文字列で表したものである．たとえば if 文の条件分岐により，一方の実行経路では “A” という文字列が出力され，他方の実行経路では “B” という文字列が出力されるとする．この場合，選択記号の | を用いて，“A | B” と表すことにより，両方の出力結果を 1 つの正規表現で表記する．同様に while 文や for 文の繰返し中に “C” という文字列が出力される場合には，繰返し記号の *

表 1 DTD での繰返し定義
Table 1 Definition of repeatable element in DTD.

a?	0 回もしくは 1 回
a*	0 回以上
a+	1 回以上

を用いて，“C* ” と表す．

依存グラフを解析することにより，文字列を出力する文（以降，出力文）に含まれる変数から，上記のような正規表現を含む出力文字列を得ることができる．ここで出力文がとりうる引数として，基本型（数値，文字列，null，undefined）およびオブジェクトの定数と変数があるが，オブジェクト（配列を含む）を引数にとった場合，ブラウザやインタプリタが行う処理の結果は動作時にのみ決定される．このため，静的な解析を行う本手法では単なる文字列として扱う．さて，解析の結果，変数の値が静的に決まる場合にはその値を，動的に決まる場合には不定値とする．なお，クラスライブラリを用いた処理が含まれる場合，クラスライブラリ内での処理は解析対象としないため，同様に不定値とする．この解析を行う際に用いられる，任意の出力文に出現する変数に対してその値の候補を計算するアルゴリズムについては付録 A.1 で述べる．

4.3 フェーズ 3：構文検証

フェーズ 3 ではフェーズ 2 で得られた出力文字列からタグ情報を解析し，フェーズ 1 で得られたタグ情報とあわせて XHTML 文書の構文検証を行う．本フェーズでは，1）正規表現で表された文字列からタグの情報を解析し，2）フェーズ 1 で得られたタグ情報とあわせて，文書全体のツリーを生成する．そして，3）このツリーについて，DTD の要素定義と検証することで構文の検証を行うという 3 つの手順を経る．以下，各手順について説明する．

4.3.1 出力文字列の解析

ここでは，フェーズ 2 で得られた出力文字列について，そこからタグ情報を得るための解析を行う．

- 基本文の処理
  XML 字句解析を行う．タグ情報をスタックに積む．
- 選択文の処理
  それまでのスタックを分岐の数だけ複製し，すべての分岐について基本文と同じ処理を行う．
- 繰返し文の処理
  繰返しが 0 回，1 回，2 回のいずれかであるとして，選択文と同様の処理を行う．

正規表現の包含関係判定問題を解決することで，検証を行うところであるが，この検証には指数時間かかることが分かっている[12]．しかし，DTD で用いられる繰返しの記述は**表 1** のように限定されている．そのため，たかだか 0，1，2 回の繰返しを試してどれでも問題なければ，3 回以上の繰返しについては試さなくてもよい．よって，ここで解析時に繰返し文を複数の選択文と同様に扱っても正確性を損なうことはない．

```
1.  スタックからルート要素開始タグを POP;
2.  P := ルート要素;
3.  C := スタックからタグ情報を POP;
4.  if (C が開始タグである) {
        P の子要素に C を追加;
        P := C;
    }
    else if (C が終了タグである) {
        if (P と要素名が不一致) エラー;
        P := P の親要素;
    }
    /* テキストおよび空要素 */
    else {
        P の子要素に C を追加;
    }
5.  if (スタックが空でない) goto 3;
```

図 3 ツリー生成アルゴリズム
Fig. 3 Algorithm of creating tree.

```
1:<html>
2:  <head><title>sample</title></head>
3:  <body>
4:    <script>
5:      var a = "good morning";
6:      var b = "good afternoon";
7:      var date = new Date();
8:      document.write("<ul>");
9:      if (date.getHour() < 12)
10:       document.write("<li>"+a+"</li>");
11:     else document.write(b);
12:     documaent.write("</ul>");
13:   </script>
14:  </body>
15:</html>
```

図 4 サンプルコード
Fig. 4 Sample code.

```
"<ul>",(("<li>","good morning","</li>")
| "good afternood"),"</ul>"
```

図 5 出力文字列
Fig. 5 Pattern of output string.

4.3.2 ツリーの生成

XHTML 構文解析で得られたタグ情報と出力文字列を解析して得られたタグ情報を組み合わせて，要素の親子関係を表したツリーを生成する．ツリーを生成するアルゴリズムを**図 3** に示す．ただし，以下，**図 4** のコードを例にして説明する．

前段階での解析結果より，本アルゴリズムへは**図 6** にあるように，2 つの場合についてそれぞれ本アルゴリズムが適用される．これは，**図 5** には 1 つの選択記号が含まれているためであり，本手順によってそれぞれツリーが生成されることとなる．1 つめの入力を

```
"<html>","<head>","<title>","sample","</title>","</head>",
"<body>","<ul>","<li>","good morning","</li>","</ul>",
"</body>","</html>"
```

```
"<html>","<head>","<title>","sample","</title>","</head>",
"<body>","<ul>","good afternoon","</ul>",
"</body>","</html>"
```

図 6 出力文字列と静的 XHTML タグの結合結果
Fig. 6 Join results of static XHTML tags and script output.

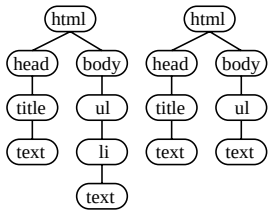

図 7 生成されたツリー
Fig. 7 Created tree.

```
1.  element := ルート要素;
2.  if (element が子要素を持っている) {
        children := element の子要素;
        definition := element の内容モデル;
        if (children が definition にマッチしない)
            DTD 違反;
        element := children;
        goto 2;
    }
```

図 8 ツリー検証アルゴリズム
Fig. 8 Algorithm of validating tree.

本アルゴリズムに適用することにより，**図 7** の左側ツリーが，また 2 つめの入力によって図 7 の右側ツリーが得られる．

4.3.3 ツリーの検証

生成された各ツリーに対して，DTD に照らした検証を行う．検証アルゴリズムを**図 8** に示す．すべてのツリーに対して検証を行い，DTD 違反がなければ，その XHTML 文書の構文は正しいという結果を示す．

もし DTD 違反が発見された場合は，その要素が XHTML 文書中のどの位置に存在しているかを特定する．もし動的コンテンツ内にその要素が含まれる場合には，依存グラフ上でその要素からグラフを逆向きにたどり，出力に関係するすべてのノードとその位置を特定する．特定した結果を DTD に違反する内容として返す．

4.4 検 証 例

ここでは，午前中に実行すると DTD 違反とはならないが，午後に実行すると違反となるような，図 4 に示すインラインスクリプトを例にして，提案する検証手法を適用した結果を示す．まず，インラインスクリプトの出力文字列を正規表現で表したものは，図 5 となる．この出力文字列を解析し，静的なタグ情報とあわせてツリーを生成すると，図 7 に示す 2 つのツリーが生成される．各ツリーに対して検証アルゴリズムを適用すると，左のツリーでは違反がなく，右のツリーでは ul 要素の子要素であるテキスト要素に DTD 違反があることが分かる．よって，このインラインスクリプトの出力結果は DTD に違反していると判定され，違反が含まれる場所を結果として得られる（**図 9**）．このように，本手法ではテストデータを用意することなくすべての実行経路について検証を行えるため，DTD 違反を容易に発見することができる．

```
Line 11: DTD violation found.
Appearance: #PCDATA
Requirement: li+
Data-Flow: Line 6, Line 11
```

図 9 検証結果例
Fig. 9 Exmaple of validation result.

5. 実 装

5.1 ツール ECMAX の概要

先に述べた検証手法を用いて，ECMAScript で記述されたインラインスクリプトを含む XHTML 文書に対する構文検証システム ECMAX の試作を行った．開発環境は以下のとおりである．

- CPU：Pentium4 2 GHz
- RAM：2 GB
- OS：FreeBSD 4.7–RELEASE
- 言語：Java（約 9,000 行）

5.2 ECMAX の構成

ECMAX の構成を**図 10** に示す．本システムは GUI，XML 解析部，ECMAScript 解析部，出力文字列作成・解析部，検証部からなる．

ユーザは GUI を通して XHTML および DTD を入力し，GUI は検証結果を表示する．

XML 解析部では，入力として XHTML 文書を受け取り，構文解析を行う．具体的には，SAX パーサを用いて XHTML 文書の構文解析を行い，タグの名前や属性，テキスト内容を得る．また，XHTML 文書中の script 要素から ECMAScript の記述を得る．さらに，DTD を構文解析して要素の定義を取得する．

ECMAScript 解析部では，入力として XML 解析部で得られたインラインスクリプトを受け取り，構文解析およびデータフロー解析を行って，抽象構文木お

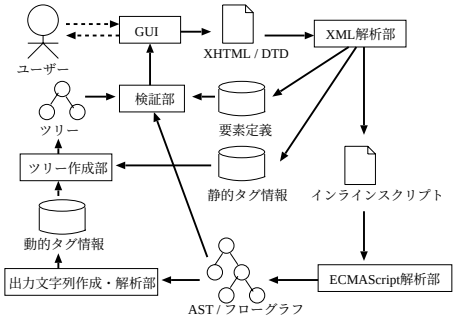

図 10 検証システムの構成
Fig. 10 Structure of validation system.

よび依存グラフを作成する．

出力文字列作成部では，インラインスクリプトの抽象構文木および依存グラフを受け取り，出力文字列を生成する．出力文字列解析部では，得られた出力文字列からタグ情報を作成する．

ツリー作成部では，XML 解析部で得られた XHTML タグ情報と，出力文字列解析部で得られたインラインスクリプトのタグ情報を受け取り，それぞれのタグ情報を統合して文書全体のツリーを作成する．

検証部ではツリーをルート要素からたどりながら，その要素の子要素が定義と一致しているか検証する．親子関係に誤りがあればそれは DTD 違反としてユーザに警告を通知し，誤りがなければ正しい文書であることをユーザに通知する．

5.3 ECMAX の実行例

図 4 の文書に対して，ECMAX を用いて検証した実行画面を**図 11** に示す．DTD 違反が発見されており，DTD 違反を起こす可能性のある実行経路が反転表示されている．

6. 評価実験

本章では，ECMAScript で記述されたスクリプトを含む XHTML 文書に対して ECMAX を用いて構文の検証を行い，その検証結果から本手法の評価を行う．

6.1 実験対象

実験対象として，Web サイト上に公開されている文書のうち，出力文を用いて HTML 文書を動的に生成しているもの 232 個を収集した．さらに変換ツールを用いて HTML から XHTML へ変換した．具体的には，XHTML1-Strict DTD 準拠の文書と XHTML1-Transitional DTD 準拠の文書をこの変換によって得た．

6.2 実験方法

すべてのサンプルデータに対して，XHTML 構文

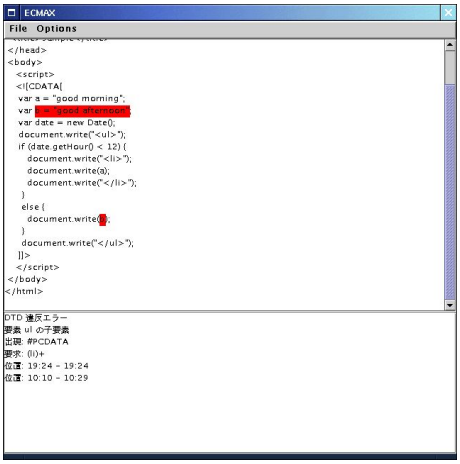

図 11 ECMAX の実行画面
Fig. 11 Screenshot of ECMAX.

チェックおよび ECMAScript 構文チェックを行った．そのうえで，インラインスクリプトの内外を含めたタグの対応づけのチェックを行った．さらに，構文に誤りがなかったサンプルデータに対して，対応する DTD（Strict DTD もしくは Transitional DTD）を用いて，DTD と文書との整合性について検証した．また，既存の XHTML 検証ツール htmllint（2001 年 9 月 11 日版）を用いて，元の HTML 文書を入力として検証を行った．

6.3 実験結果

実験結果を**表 2** に示す．括弧内の数値は，不定値を XHTML のタグがいっさい含まれていないテキストと見なして処理を行った部分に何らかの違反が含まれていた文書，あるいは，違反なしという結果を得た文書のうち，そのような不定値が含まれていた文書の数，である．さらに htmllint では違反が検出されなかった文書を ECMAX で検証した結果は**表 3** のとおりである．

なお，表 2 において，「XHTML 構文違反」はスクリプト以外の部分に閉じた構文違反があり図 10 で述べた XML 解析部が違反を検出したものである．「スクリプト構文違反」はスクリプト自体に ECMAScript 構文エラーがあり，ECMAScript 解析部が違反を検出したものである．「スクリプト内 XHTML 構文違反」はスクリプトの出力文字列内で閉じた XHTML 構文違反があり，出力文字列生成・解析部が違反を検出したものである．「スクリプト内タグを含むタグ付け違反」は，スクリプトの出力を含んだ文書全体が XML 文書として（DTD によらず）構文違反を含んでおり，ツリー作成部が違反を検出したものである．「DTD 違

表 2 ツールによって検出された XHTML 文書数
Table 2 Number of XHTML documents found by tools.

ツール	ECMAX		htmllint	
DTD	Strict	Transitional	Strict	Transitional
XHTML 構文違反		10		10
スクリプト構文違反		48		—
スクリプト内 XHTML 構文違反		43 (14)		—
スクリプト内タグを含むタグ付け違反		18 (14)		—
DTD 違反	113 (35)	65 (12)	153	8
違反なし	0	48 (23)	69	214
合計		232 (63)		232

表 3 htmllint で無違反だった文書の ECMAX での検証結果
Table 3 Validation result of judged "no error" by htmllint.

DTD	Strict	Transitional
スクリプト構文違反	13	46
スクリプト内 XHTML 構文違反	13	42
スクリプト内タグを含むタグ付け違反	9	16%
hline DTD 違反	34	63
違反なし	0	47
合計	69	214

反」は，上記のすべての違反がまったく含まれていないが，DTD に照らして検証を行った場合に違反があるもので，検証部によって違反を検出したもの，である．検証作業はこれら 5 つについて順番に行われ，どこか 1 カ所で違反が発見された場合には，そこで解析を終了している．

6.4 考　　察

図 2 および図 3 から，ECMAX に htmllint が発見できなかった多くの違反を発見できていることが分かる．また，ECMAX によって Transional DTD による検証の結果「違反なし」とされた 48 の文書のうち，47 については htmllint も「違反なし」という検証結果を出力しているが，残る 1 つについては htmllint は DTD 違反であると誤って検出した．これは，htmllint がインラインスクリプトの出力を考慮できなかったことによるものである．本手法によるインラインスクリプトの出力結果を考慮した検証によって，より正確な検証結果を得ることができると考えられる．

また，不定値を含む文書は全体の 3 割程度（232 中 63）であり，その他の文書はタグを静的に記述し，メッセージを動的に生成するという文書であった．したがって，本手法の適用範囲は広範囲に及ぶものであり，本手法の有効性は高いと考えられる．

しかしながら，本手法は静的な解析を行っているため，ECMAScript で記述されたインラインスクリプトの出力へ，動作時にのみ決定される文字列が含まれており，かつその部分に XHTML 構文要素と見なされる文字列がある場合には，構文検証を正しく行えない．具体的には，動作時にのみ決定される文字列に含まれている XHTML 構文要素を含めて，全体として正しい構文であるような XHTML 文書を，本手法では誤りを含む文書として判断されてしまう．構文検証の精度を向上させるためには，本手法で提案する静的な解析に加え，実行時の情報を用いた解析が必要であろう．

7. ま と め

本研究では，ECMAScript で記述されたインラインスクリプトを含む XHTML 文書を対象として，DTD に基づいた構文であるかを検証するための手法を提案した．本手法は，スクリプトの出力文に対してデータフロー解析を行い，出力文字列を正規表現を含む文字列で表したうえで，その文字列の検証を行うものである．また，本手法に基づく検証システムを試作し，インターネット上に存在するインラインスクリプトを含む XHTML 文書に対して検証を行い，本手法の評価を行った．その結果，動的コンテンツの内容を考慮した解析によって，文書の構文間違いなどをより多く発見できることが分かった．

今後の課題としては，ECMAScript 言語が提供するライブラリへの対応，変数のエイリアス解析，XML Schema への対応などがある．

参 考 文 献

1) W3 Consortium: XHTML 1.0: The Extensible HyperText Markup Language, A Reformulation of HTML 4.0 in XML 1.0. Working Draft (1999). http://www.w3.org/TR/1999.
2) 鷲尾和則，松下　誠，井上克郎：JavaScript を含んだ HTML 文書に対するデータフロー解析を用いた構文検証手法の提案，電子情報通信学会研究技術報告，Vol.SS2002-22, pp.13–18 (2002).
3) W3 Consortium: HTML 4.01 Specification (1999). http://www.w3.org/TR/1999/REC-

html401-19991224
4) ECMA: ECMAScript Language Specification (1999). http://www.ecma.ch/stand/ecma-262.htm
5) 鷲尾和則，松下 誠，井上克郎：JavaScript を含む HTML 文書の妥当性検証手法の提案，電子情報通信学会総合大会講演論文集，Vol.D-3-5, p.31 (2002).
6) Brabrand, C., Møller, A. and Schwartzbach, M.: Static validation of dynamically generated HTML, *Proc. Workshop on Program Analysis for Software Tools and Engineering* (*PASTE 2001*), Snowbird, Utah, USA (2001).
7) W3 Consortium: Document Object Model (DOM) Level 1 Specification, Version 1.0. Recommendation (1998). http://www.w3.org/TR/1998
8) Kempa, M. and Linnemann, V.: V-DOM and P-XML — Towards Valid XML Applications (2002).
9) Meijer, E. and Shields, M.: XMλ: A Functional Language for Constructing and Manipulating XML Documents (1999). (Draft).
10) Hosoya, H. and Pierce, B.C.: XDuce: A Typed XML Processing Language, *Int'l Workshop on the Web and Databases* (*WebDB'2000*), Dallas, Texas (2000).
11) Horwitz, S. and Reps, T.: The Use of Program Dependence Graphs in Software Engineering, *14th International Conference on Software Engineering*, Melbourne, Australia (1992).
12) Seidl, H.: Deciding Equivalence of Finite Tree Automata, *SIAM Journal on Computing*, Vol.19, No.3, pp.424–437 (1990).

付 録

A.1 変数値候補の計算アルゴリズム

A.1.1 概 要

依存グラフから変数値の候補を計算するアルゴリズム（**図 12**）について説明する．本アルゴリズムは，ECMAScript プログラムに含まれる変数のうち，プログラム実行の際動的に決定される変数（オブジェクト）以外を対象として，任意の文中におけるある変数のとりうる値の候補を計算するものである．

依存グラフから変数値の候補を計算するアルゴリズム（図 12）について説明する．本アルゴリズムは，いくつかの制約を加えた ECMAScript プログラムを対象として，任意の文中におけるある変数のとりうる値の候補を計算するものである．

本アルゴリズムでは最初に，たとえば**図 13** のような ECMAScript のソースコードから，**図 14** のような依存グラフを構築する（図 12 の 1）．依存グラフの頂点は文または式であり，有向辺は始点の次に終点が実行されうることを示す．値を調べたい文に対応する頂点を対象頂点，値を調べたい変数を対象変数と呼ぶ．

```
1.  スクリプトを構文解析，意味解析;
    program := スクリプトの抽象構文木;
2.  node := program のルートノード;
3.  if (node = 繰返し制御構造文) 繰返し記号 "*" を追加;
    else if (node = 選択制御構造文) 選択記号 "|" を追加;
    else if (node = 出力文) {
        引数に対してデータフロー解析，値を求める;
        if (変数の値が動的に決まる) 値 := 不定値;
        else                        値 := 定数値;
        値を追加;
    }
4.  if (node が子ノードを持っている) {
        node := node の子ノード;
        goto 3;
    }
```

図 12 変数値候補の計算アルゴリズム
Fig. 12 Algorithm of creating pattern.

```
var a=10;
for (var i=0; i<10; i+=1) {
  a += i;
}
document.writeln(a);
```

図 13 ソースコードの例
Fig. 13 Example of source code.

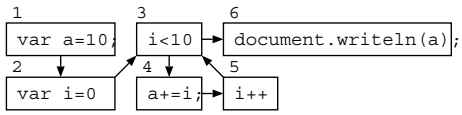

図 14 図 13 の依存グラフ
Fig. 14 Flow graph of example code.

対象頂点を始点とし（図 12 の 2），対象変数への代入が見付かるまで，有向辺を逆向きに辿る（図 12 の 4）．見付かった代入文の右辺に変数がある場合は，その変数の値を再帰的に調べ，右辺のとりうる値を決定する（図 12 の 3）．この手順だけでは，依存グラフに閉路がある場合に，無限に再帰してしまう場合がある．これを避けるために，調査中の頂点と変数の組を記憶しておき，二重の探索を避ける．

以下，対象頂点 n を実行した直後の対象変数 x の値の候補を x^n と表す．値の候補は，具体的な値（1, "str"，true など）と，型の決まった不定値（不定値（数値），不定値（文字列）など）の集合で示される．探索中の対象頂点と対象変数の組を記憶しておく大域変数を Q とする．x^n を求めるためのアルゴリズムは，以下のとおりである．

(1) Q に，対象頂点と対象変数の組 (n, x) が含まれているか調べる．含まれている場合は，x^n を展開せず，そのまま返す．含まれていない場合

は，(n, x) を Q に追加する．

(2) n の種類によって，以下の処理を行う．

対象変数への代入文でない場合 n に入ってくる辺の始点の集合を N_{in} として，$r{=}\cap_{m\in N_{in}}\mathrm{x}^m$ とする．

対象変数への代入文で，右辺が定数の場合 $r =$ { 右辺の値 } とする

対象変数への代入文で，右辺に変数がある場合 右辺に含まれる変数それぞれの値候補を計算する．右辺の式に，得られた値候補を代入し，r に加える．r の要素に x^n を含む式があった場合は，その式に x^n を含まない式を代入し，r に加えることを，r の要素が増えなくなるまで繰り返す．ただし，x^n の要素として x^n 自身が含まれていた場合には，x^n の値のうち，具体的な値として求まっている物を代入したうえで，その演算結果の型のみを用いて,「不定値（得られた型)」とする．

(3) Q から (n, x) を削除し，r を返す．

このようにして求められた，出力文に含まれる引数の値について，抽象構文木の情報から，それぞれの出力文が if 文や while 文などの制御構造下にある場合，正規表現で用いられる選択記号 (|) や繰返し記号 (*) を付加する．選択記号 (|) を付加する制御構造は，if 文，switch 文である．また，繰返し記号 (*) を付加する制御構造は，while 文，for 文である．また，連続するそれぞれの出力文字列の間には連結記号 (,) を挿入する．

A.1.2 具体的な計算例

ここでは，図 13 の最後の行で参照されている変数 a について，その候補 a^6 を具体的に求める手順について説明する．

まず $Q = \{(6,\mathrm{a})\}$ としたうえで，a^6 を計算する，依存グラフを逆向きにたどることにより，$\{\mathrm{a}^1, \mathrm{a}^4\}$ であることが分かる．ここで，$\mathrm{a}^1 = \{10\}$ であり，$\mathrm{a}^4 = \{\mathrm{a}^3 + \mathrm{i}^3\}$ であることから，次に Q に $(4, \mathrm{a})$ を追加したうえで a^3 を求める．$\mathrm{a}^3 = \{\mathrm{a}^1, \mathrm{a}^4\}$ であるが，ここで $(4, \mathrm{a})$ が Q に含まれていることと，すでに a^1 の値が求まっていることから，$\mathrm{a}^3 = \{10, \mathrm{a}^4\}$ として a^3 の計算を終える．

次に i^3 を求める．これは i^2，i^5 となるが，ここで i^2 は $\{0\}$ であり，$\mathrm{i}^5 = \{\mathrm{i}^4 + 1\}$ であることから，Q へ $(5, \mathrm{i})$ を加えて i^4 の値を求めることとする．i^4 の値は $\{\mathrm{i}^2, \mathrm{i}^5\}$ であるが，$(5, \mathrm{i})$ は Q に含まれていたためここで計算を終了し，既知の結果とあわせて，$\mathrm{i}^5 = \{1, \mathrm{i}^5 + 1\}$ となる．このとき，左辺の i^5 は消去できて，$\mathrm{i}^5 = \{1,$ 不定値 (数字)$\}$ となる．この結果，i^3 の値はこれまでの結果をあわせて $\{0, 1,$ 不定値 (数字)$\}$ であることが分かる．

a^3 と i^3 の値が求められたので，a^4 の計算を行うことができる．a^3 には 2 つ，i^3 には 3 つの要素があるため，6 つの足し算を行い，その結果を a^4 とすることとなる．ただしこのとき，不定値（数字）へ定数の演算を行っても，やはり不定値（数字）であること，複数の同じ要素は 1 つにまとめることができることから，$\{10, 11,$ 不定値 (数字), $\mathrm{a}^4, \mathrm{a}^4 + 1\}$ を得る．a^4 の計算結果中に a^4 が再度出現したため，上記の i^5 のときと同様に a^4 を消去すると，$\{10, 11,$ 不定値 (数字)$\}$ が得られる．

以上の結果から，$\mathrm{a}^6 = \{\mathrm{a}^1, \mathrm{a}^4\} = \{10, 11,$ 不定値 (数字)$\}$ を得ることができ，これが求める候補となる．

(平成 15 年 10 月 27 日受付)
(平成 16 年 6 月 8 日採録)

松下 誠（正会員）

平成 5 年大阪大学基礎工学部情報科学科卒業．平成 10 年同大学大学院博士後期課程退学．同年同大学基礎工学研究科助手．平成 14 年同大学情報科学研究科助手．博士（工学）．ソフトウェア開発環境，ソフトウェア開発プロセス，オープンソース開発の研究に従事．

鷲尾 和則

平成 13 年大阪大学基礎工学部情報科学科退学．平成 15 年同大学大学院情報科学研究科卒業．同年三菱電機先端技術総合研究所．在学時，XHTML 文書検証の研究に従事．

井上 克郎（正会員）

昭和 54 年大阪大学基礎工学部情報科学科卒業．昭和 59 年同大学大学院博士課程修了．同年同大学基礎工学部情報科学科助手．昭和 59 年〜61 年ハワイ大学マノア校情報工学科助教授．平成元年大阪大学基礎工学部情報科学科講師．平成 3 年同学科助教授．平成 7 年同学科教授．工学博士．ソフトウェア工学の研究に従事．