修士学位論文

題目

インラインスクリプトに対するデータフロー解析を用いた XHTML 文書の構文検証

指導教官
井上 克郎 教授

報告者
鷲尾 和則

平成 15 年 2 月 12 日

大阪大学 大学院基礎工学研究科
情報数理系専攻 ソフトウェア科学分野

平成 14 年度 修士学位論文

インラインスクリプトに対するデータフロー解析を用いた XHTML 文書の構文検証

鷲尾 和則

内容梗概

インターネットで公開されている文書の多くは、HTML や XHTML などのマークアップ言語を用いて記述されている。これらの文書は、マークアップ言語によって文書の体裁や構造などを記述するだけでなく、JavaScript や PHP 等といったスクリプトを用いて内容が動的に変化することも多い。記述された内容が正しく閲覧できることを確認するために、文書に対する構文検証を行う手法が知られている。しかし、従来の検証方法では、スクリプトによって生成される内容に対する検証は行われておらず、構文に誤りがないと判定された文書であっても実際には誤りを含むことがあった。

本研究では、インラインスクリプトを含んだ XHTML 文書を対象として、構文定義に基づいたタグ付けが行われているかをデータフロー解析を用いて検証する手法を提案する。本手法ではまず、インラインスクリプト内の出力文に含まれる変数に対してデータフロー解析を行い、XHTML で記述された部分の構文解析結果とあわせて出力される文字列のパターンを生成する。その後、パターンに対して XHTML の構文に照らして検証を行う。また、本手法を用いた構文解析ツールの実装を行い、インラインスクリプトを含む実際の XHTML 文書をツールに適用することによっ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構文の誤りを多数検出することができ，本手法の有効性について確かめることができた.

主な用語

XHTML (eXtensible Hyper Text Markup Language)
ECMAScript
データフロー解析 (Data Flow Analysis)
構文検証 (Validation)

目 次

1 まえがき

近年，インフラの整備や情報技術の発展にともない，インターネットが広く普及し，膨大な量にのぼるホームページなどの電子文書が作成されている．その多くは Hyper-Text Markup Language (HTML) [1] というマークアップ言語によって記述されている．HTML 文書は，タグという記号で区切られた要素という情報の単位をいくつか並べて記述することで文書全体が構成されている．

そのタグ付け規則は，HTML 文書の Document Type Definition (DTD: 文書型定義) において定義されており，すべての HTML 文書は DTD に従って記述することが求められる．もし DTD に違反したタグ付けが行われた場合，文書交換に支障を来す他，ウェブブラウザで正しく表示されないなどの問題が発生する．そのため，HTML 構文が DTD にしたがって正しく記述されているかどうかを検証する必要がある．

一方，HTML 文書には JavaScript [2, 21] JAVA Server Pages (JSP) [16], PHP [17] などの言語で書かれたインラインスクリプトを文書内に挿入することができる．スクリプトはクライアントまたはサーバーによって実行され，任意の HTML 文書の断片を出力することができる．Microsoft Internet Explorer や Netscape Navigator などの主なウェブブラウザが JavaScript (あるいは JScript [18]) の表示をサポートしていることから，JavaScript は広く使われるようになった．しかし，JavaScript の構文は HTML の文法とは独立しているため，しばしば誤った構文の HTML 記述を出力するスクリプトを書いてしまうことがある．

正しい HTML を記述するという立場から，HTML 文書を検証するために，現在，HTML 文書の構文を検証する多くのツールが存在する．しかしそれらのツールは JavaScript によって書かれたスクリプトの内容を無視するため，スクリプトの間違いを発見できない．そこで，JavaScript で書かれたスクリプトの内容を考慮して HTML 文書の構文の検証を行う必要がある．

そこで，本研究では，インラインスクリプトとして ECMAScript [5] を含んだ， HTML 文書の拡張である XHTML (eXtensible Hyper-Text Markup Language) [4, 23, 24, 22] 文書について，その文書が DTD 定義に基づいた構文であるかを検証するための手法を提案する．

本手法では，スクリプトに対して，出力文に含まれる変数のデータフロー解析を行い，構文解析の結果と合わせて，出力される文字列を正規表現を用いてパターン化する．そしてそのパターンについて文書のツリーを生成し，その構文の検証を行う．

また，本手法をシステムに実装し，インラインスクリプトを含んだ実際の XHTML 文書に対して，検証を行ない，評価を行なった．

以降，第 2 章では XHTML と ECMAScript，第 3 章では検証手法について説明する．第 4 章では試作システム「ECMAX」を紹介する．第 5 章では実際にインラインスクリプトを含

む XHTML 文書に対して検証を行ない，本手法を評価した．第 6 章では本論文のまとめと今後の課題について述べる．

2　構造化文書とスクリプト

本章では，構造化文書を記述するためのマークアップ言語およびスクリプトに関する説明を述べる．また，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

XHTML は，HTML の仕様と同じものを eXtensible Markup Language (XML) [3, 23, 22] を用いて改めて仕様化した言語で，タグの省略が不可能である等，厳密な言語定義がなされており，今後のウェブページ作成の標準となりうる言語である．

2.1.1　HTML

HTML はウェブページ作成のために，World Wide Web Consortium (W3C) が標準化したマークアップ言語である．現在，多くのウェブページが HTML で作成されている．

しかしながら，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する．

2.1.2　XML

XML (eXtensible Markup Lanugage) は W3C が制定した「拡張可能なマークアップ言語」である．

XML は他の言語を定義するためのメタ言語としても用いられる．XML で HTML を定義したものが XHTML である．

- 要素

  XML では文書を要素 (element) という単位で構成する．それぞれの要素は，タグとよばれる記号 (< >) で区別する．要素をならべて書いたり，要素の中に別の要素を挿入することで，文書を構造化して書くことができる．

XML 文書の構成を図 1 に示す．

- XML 宣言

  XML のヴァージョンや文字コードを指定する．

- DTD

  DTD は Document Type Definition の略であり，日本語では「文書型定義」と訳される．文書を構造化するための設計図とも言うべきものである．この DTD には，要素

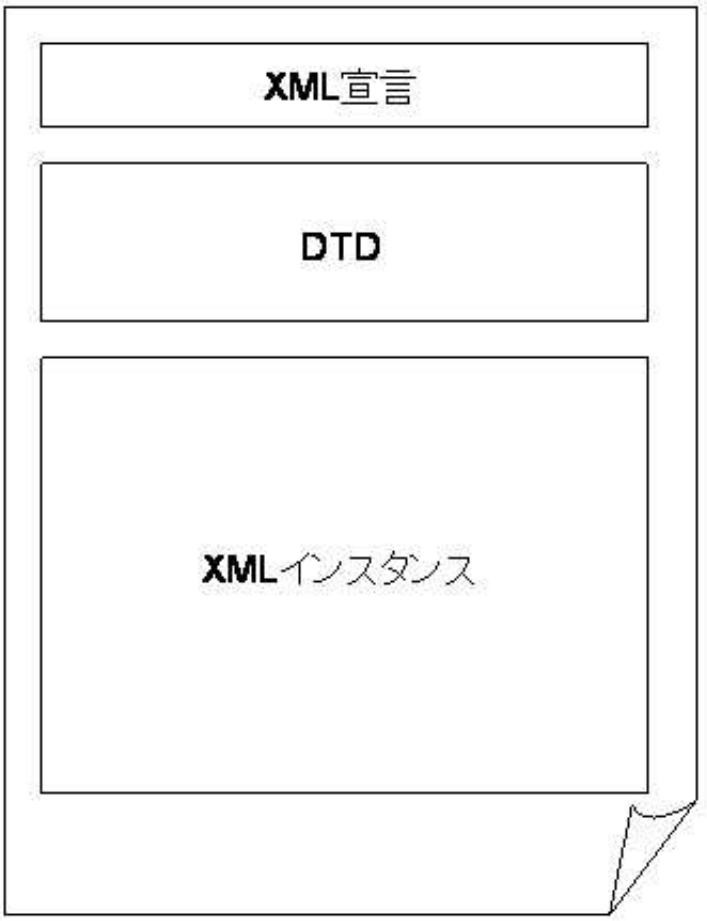

図 1: XML 文書の構造

の定義 (要素の名前，子要素の出現順序や出現回数，属性リストの定義) を記述する．このうち，構造化する上で重要な定義は，子要素の出現順序や出現回数で，内容モデルと呼ばれる．内容モデルによって，親子関係が定義されているため，内容モデルに違反した文書は正しく構造化されていないということが判定できる．

DTD の例を図 2 に示す．1 行目の 文書型宣言により，この文書のルート要素 (memo) を指定している．2 行目以降では，それぞれの要素について，その定義を宣言している．例えば，2 行目では，memo 要素の子要素として，head 要素，body 要素がこの順番で出現することを示している．また，4 行目の title 要素の宣言では，子要素が #PCDATA となっているが，これはテキスト要素を意味している．このようにして，要素の親子

関係を定義する．

```
<!DOCTYPE memo [
<!ELEMENT memo (head, body)>
<!ELEMENT head (title)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

図 2: DTD の例

- XML インスタンス
  XML インスタンスには，実際の文書，つまり要素を構造化して記述する．
- Well-formed XML Document と Valid XML Document
  XML はその処理上，2 つの文書に分類される．Well-fromed XML Document (整形式 XML 文書) と Valid XML Document (検証済 XML 文書) である．
  Well-formed XML Document は，開始タグと終了タグの整合が取れており，入れ子関係なども正しく記述された文書である．すなわち，XML 文書として最低の条件を満たしている文書ということになる．
  一方，Valid XML Document は，Well-formed XML Document の条件に加えて，要素の名前，親子関係などが DTD の定義に従っている必要がある．
  Well-formed XML Document と Valid XML Document の関係を表 1 に示す．

表 1: Well-formed XML Document と Valid XML Document の関係

文書の種類	XML 宣言	DTD	XML インスタンス
Well-formed XML Document	△	△	○
Valid XML Document	△	○	○

○: 必須
△: なくてもよい

2.2 ECMAScript

ECMAScript は，JavaScript をもとに European Computer Manufacturer Association (ECMA: 欧州電子計算機工業会) によって標準化されたスクリプト記述言語である．ブラウザに固有

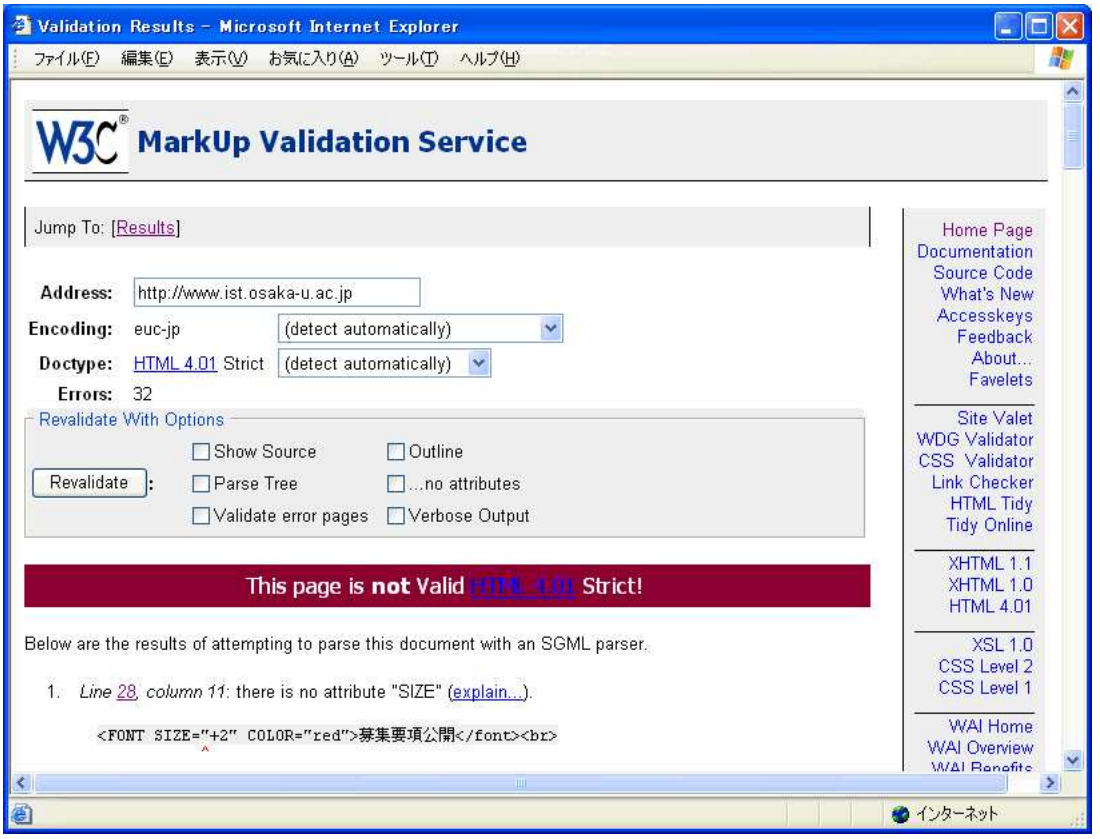

図 3: HTML Validator

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

HTML 文書や XHTML 文書，XML 文書などの構造化文書は，その構文の正しさが重要となる．誤って構造化された文書は，その交換や処理に障害を及ぼすからである．ゆえに，その構文を検証するシステムが数多く存在する．例えば，W3C のオフィシャルサイト (http://www.w3.org/) には，HTML 文書や XHTML 文書の構文を検証するページがある．(図 3)

しかしながら，これら既存の検証システムでは，スクリプトの内容は単なる文字列データと見なされ，その内容までは検証しない．したがって，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では判断ができないのである．よって，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，入力として適当なデータを与えてインタープリタを介すれば，

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．([25]) しかし，テストデータによっては実行結果が変わり，検証結果も変わる可能性がある．よって全ての実行経路を通るようにテストデータを用意しなければならないが，それは非現実的である．

そこで，本研究では，スクリプトに対して，出力文に含まれる変数のデータフロー解析を行い，構文解析の結果と合わせて，出力される文字列を正規表現を用いてパターン化する．そしてそのパターンについて文書のツリーを生成し，その構文の検証を行うという手法を提案する．

3　検証手法

提案する検証手法の入出力は以下の通りである．

入力: XHTML 文書および DTD
出力: 検証結果

具体的には XHTML 文書中にある ECMAScript が出力する文字列をパターンで表し，そのパターンを検証する．この手法により，動的に変わりうる出力文字列をパターン化して検

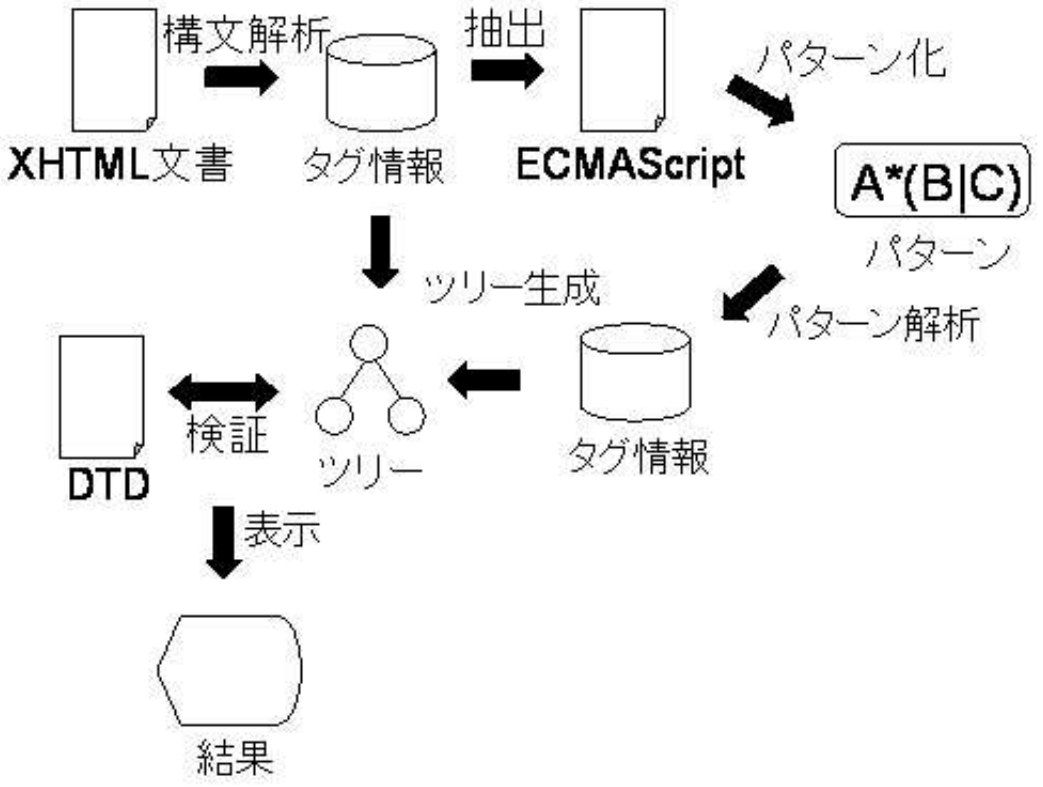

図 4: 検証手法の流れ

この手法は大きく分けて次の 3 つの段階からなる．

- フェーズ 1: XHTML 文書解析
  まずは，対象となる XHTML 文書について構文解析を行い，タグ情報を格納し，同時に XHTML 記述と ECMAScript 部分を分離する．
- フェーズ 2: スクリプト解析
  次に，ECMAScript で書かれたスクリプトの解析を行う．これは構文解析及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．それらの結果を元に，繰り返し (*) や選択 (|) などの正規表現を用いて出力文字列をパターンとして表現する．
- フェーズ 3: 構文検証
  最後に，フェーズ 2 で得られたパターンを解析し，タグ情報を生成する．そしてフェーズ 1 で得られたタグ情報と合わせてツリーを生成し，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．

3.1 フェーズ 1: XHTML 文書解析

フェーズ 1 では，XHTML 文書の構文解析を行う．ここで得られるスクリプトは次のフェーズ 2 へ渡す．

解析にはイベント駆動型の XML パーサを用いる．このパーサーを用いて XHTML 文書を構文解析すると，タグ (その要素名や属性リスト) あるいはテキスト要素の情報が取得できる．

3.2 フェーズ 2: スクリプト解析

フェーズ 2 では，スクリプトの解析を行う．本フェーズは，1) ECMAScript で書かれたスクリプトの構文解析を行い，2) 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．3) 先ほどの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で表すという 3 つの手順で行われる．この正規表現は次のフェーズ 3 で用いられる．

3.2.1 構文解析

ECMAScript の構文定義 [5] をもとにスクリプトの構文解析を行い，スクリプトの抽象構文木を作成する．そして，if 文や while 文などの制御構造を判別する．これは後述のパターン化において必要となる．

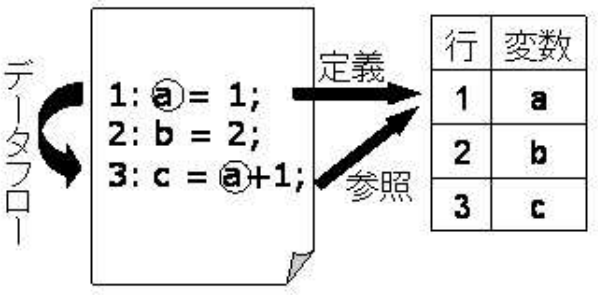

図 5: データフロー解析

3.2.2 データフロー解析

データフロー解析は，参照された変数がプログラム (スクリプト) 中のどの位置で定義されているのかを解析するものである．この解析を行うため，まずスクリプトに対して構文解析および意味解析を行う．そして，スクリプトの抽象構文木において，ルート (根) から順に探索を行ない，変数が定義されれば変数表にその変数名や位置などの情報を追加する．変数が参照されれば変数表にその変数があるか確かめる (図 5).

これにより変数および定義されている位置と参照されている位置について記したリストを得ることができる．この際，変数のスコープなどの問題に注意する必要がある．このリストを元に，データフローをグラフ化したフローグラフを作成する．

これらの抽象構文木およびフローグラフは，次ステップの出力文字列のパターン化に使用される．

3.2.3 出力文字列のパターン化

スクリプト解析の最終段階は，出力文字列を正規表現で表し，パターン化することである．ここでパターンについて説明する．例えば 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．その両方の出力をまとめて記述するために，正規表現を用いる．この場合，選択記号の | を用いて，”A | B” というパターンとなる．同様に while 文や for 文の繰り返しについては，繰り返し記号の * を用いて，例えば ”C*” というパターンで表す．

このパターン化は，出力文に含まれる変数についてデータフロー解析を行い，変数の処理を解析することで行える．ところで出力文の引数としては，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を取ることができるが，オブジェクト (配列を

含む) を引数に取った場合，ブラウザやインタープリタが行う処理が不定なのでここでは省略する．つまり，出力文に含まれる数値，文字列の変数に対してデータフロー解析を行い，変数処理の流れ，つまり一連の変数の定義・参照関係の情報を得る．

そして，データフロー解析後，変数の処理の中で，その値が静的に定まればその値を変数の値として用いる．しかし，静的に定まらない，すなわち動的に定まる場合，その変数の値は不定値 (Unknown) とし，その変数の型で表すことにする．また，変数の処理の中でクラスライブラリのメソッドを用いた処理があった場合，その変数の値は同じく不定値 (Unknown) とする．

＜変数値候補の計算アルゴリズム＞

データフローグラフから変数値の候補を計算するアルゴリズムについて説明する．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取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，ソースコード (図 6) から，フロー図 (図 7) を構築する．フロー図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．

対象頂点を始点とし，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．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，右辺の取り得る値を決定する．この手順だけでは，フロー図に回路がある場合に，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 等) と，型の決まった不定値 (不定値 (数値), 不定値 (文字列) 等) 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 を求めるための具体的な処理は，以下の通りである．

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=$\cap_{m \in N_{\mathrm{in}}} \mathrm{x}^m$ 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

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

図 6: ソースコードの例

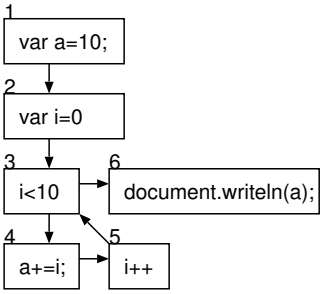

図 7: 図 6 のフロー図

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．

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，出力文に含まれる変数の値について，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 (|) や繰り返し記号 (*) 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 (図 8)．また，繰り返し記号 (*) を付加する制御構造は，while 文，for 文である (図 9)．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 (図 10)．

以上をまとめると，図 11 のアルゴリズムとなる．

3.3 フェーズ 3: 構文検証

フェーズ 3 ではフェーズ 2 で得られたパターンからタグの情報を解析し，フェーズ 1 で得られたタグ情報と合わせて 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合わせて，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 の親子関係を正

```
if (条件式) document.write("a");
else if (条件式) document.write("b");
...
else document.write("c");
```

⇓

```
( "a" | "b" | "c" )
```

図 8: 選択記号の付加

```
while (条件式) {
  document.write("a");
}
```

⇓

```
( "a" )*
```

図 9: 繰返し記号の付加

```
document.write("a");
document.write("b");
document.write("c");
```

⇓

```
( "a", "b", "c" )
```

図 10: 連続記号の付加

規表現で表した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，3 つの手順について説明する．

```
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) * 追加;
    else if (node = 選択制御構造文) | 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 (動的に定まる) 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 11: パターン作成アルゴリズム

3.3.1 パターン解析

ここでは，フェーズ 2 で得られたパターンについて，そこからタグ情報を得るための解析を行なう．

- 基本文の処理
  そのまま XML 字句解析を行なう．

- 選択文の処理
  それまでのタグ情報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なう．

- 繰返し文の処理
  繰返しの回数を不定とし， 0 回，1 回, 2 回の繰返しの場合について選択文と同じ処理を行なう．

繰返し文の繰返し回数を 0 回，1 回，2 回の 3 種類に特定する理由を説明する．DTD で用いられる繰返しの定義は表 2 のようになる．

表 2: DTD での繰返し定義

a?	0 回 もしくは 1 回
a*	0 回以上
a+	1 回以上

この定義から，繰返し部分を 2 回まで調べて，違反がなければ 3 回以上調べても違反は起こらないということを意味している．よって，繰返し文の繰返し回数は 2 回までの場合について調べることで検証できる．

本来であれば，繰返し回数を特定せず，正規表現の包含関係判定問題を解決することで，検証を行なうところであるが，この検証には指数時間かかることが分かっている．しかし，検証の高速化ならびに簡単化のためのこのような手法を取った．また，現状ではこの手法で問題となることは起こっていない．

3.3.2 ツリーの生成

ここでは，XHTML 構文解析で得られたタグ情報と，パターンを解析して得られたタグ情報を合わせて，要素の親子関係を表したツリーを生成する．これらのタグ情報は出現順にスタックに格納されている．ツリーを生成するアルゴリズムを図 12 に示す．このようにして生成されたツリーは，次の検証作業へと渡される．

3.3.3 ツリーの検証

ここでは，先ほど生成されたツリーに対して検証を行なう．検証アルゴリズムを図 13 に示す．すべてのツリーに対して検証を行ない，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．

3.4 検証例

図 14 のようなサンプルコードについて，検証例を見てみる．

このサンプルコードのスクリプトの出力文字列パターンは，図 15 のようになる．

出力文字列パターンを解析し，静的なタグ情報と合わせてツリーを生成すると，図 16 および図 17 の 2 つのツリーができる．

それぞれのツリーについて検証アルゴリズムに基づいて検証を行なう．図 16 のツリーでは違反はない．しかし，図 17 のツリーにおいて，ul 要素の子要素であるテキスト要素がその定義

```
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 (開始タグ) {
        P の子要素に C を追加;
        P := C;
    }
    else if (終了タグ) {
        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 12: ツリー生成アルゴリズム

```
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 := DTD における 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 13: ツリー検証アルゴリズム

```
<!ELEMENT ul (li)+>
```

に違反する．ゆえにこのツリーは DTD に違反している．したがって，このスクリプトを実行すると， DTD 違反となる経路 (危険なパス) が存在する．

例えば，午前中にこのスクリプトを実行しても，エラーは見つからないが，午後，スクリプトを実行するとエラーが起こる．このように，テストデータをいくつも用意しなくとも，あるいは実行環境をすべて整えなくとも，すべての実行経路について検証ができるため，エラーを容易に発見することができる．

```
<html>
  <head>
    <title>sample</title>
  </head>
  <body>
    <script>
      <![CDATA[
        var a = "good morning";
        var b = "good afternoon";
        var date = new Date();
        document.write("<ul>");
        if (date.getHour() < 12) {
          document.write("<li>");
          document.write(a);
          document.write("</li>");
        }
        else {
          document.write(b);
        }
        document.write("</ul>");
      ]]>
    </script>
  </body>
</html>
```

図 14: サンプルコード

“ ”, ((“ ”,”good morning”, “ ”) | “good afternood”), “ ”

図 15: 出力文字列パターン

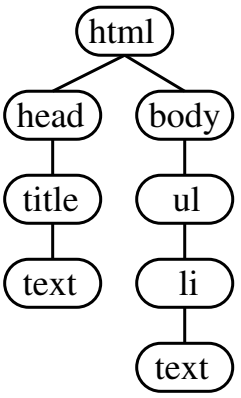

図 16: 生成されたツリー その 1

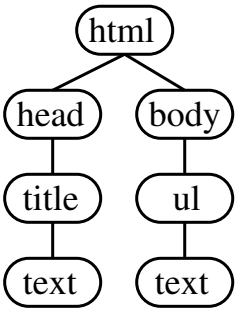

図 17: 生成されたツリー その 2

4 実装

4.1 ツールの概要

先に述べた検証手法に基づいた ECMAScript で記述されたスクリプトを内部に含む XHTML 文書に対する構文検証システム「ECMAX (ECMAscript in Xhtml document validation system)」の試作を行なった．開発環境は以下の通りである．

- CPU: Pentium4 2GHz
- RAM: 2GB
- OS: FreeBSD 4.7
- 言語: JAVA (JDK 1.3.1)
  コード量: 約 9,000 行
  使用ライブラリ
  - JAXP (Java API for Xml Processing)
  - jakarta-oro
  - DTD Parser

4.2 ツールの構成

本システムの構成を図 18 に示す．この図から分かるように，本システムは XML 解析部，DTD 解析部，ECMAScript 解析部，パターン作成·解析部，検証部から成る．

- XML 構文解析部

  XML 解析部では，入力として XHTML 文書を受け取る．そして，XHTML 文書を読み込んで構文解析を行なう．実装にはイベント駆動型の SAX パーサを用いた．解析を行なうと，タグが現れる度にその要素名や属性，テキストであればその内容を受け取る．これにより，タグの情報を生成する．また，スクリプト要素から ECMAScript の記述を得る．

- DTD 解析部

  DTD 解析部では，入力として XHTML の DTD (xhtml1-strict.dtd, xhtml1-transitional.dtd, xhtml1-frameset.dtd のいずれか) を受け取る．そして，DTD を構文解析して，要素の定義 (親要素とその子要素の内容モデル) を取得し，要素名をキー，その内容モデルをハッシュとしてハッシュテーブルを作成する．

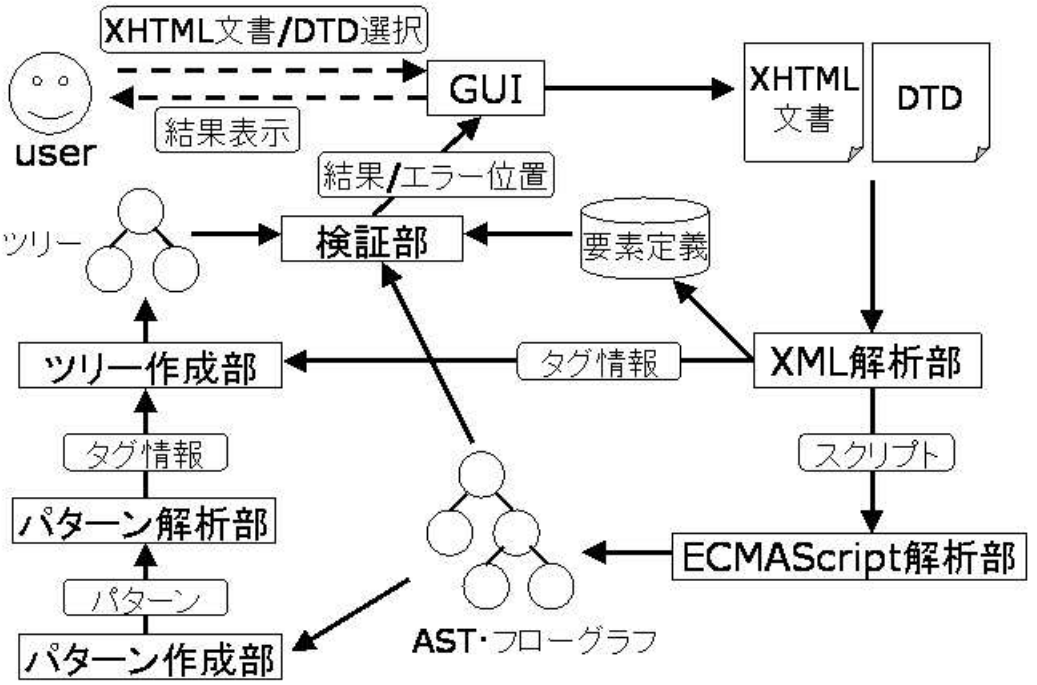

図 18: 検証システムの構成

- ECMAScript 解析部

  ECMAscript 解析部では，入力として XML 解析部で得られたスクリプト記述を受け取る．受け取ったスクリプトに対して，構文解析およびデータフロー解析を行ない，抽象構文木およびフローグラフを作成する．

- パターン作成 ·解析部

  パターン作成部では，入力として ECMAScrip 解析部で得られた，スクリプトの抽象構文木および，フローグラフを受け取る．そして，スクリプトの出力文のデータフローから，出力文字列のパターンを生成する．
  パターン解析部では，入力として作成部で得られたパターンを受け取る．そして，パターンからタグ情報を作成する．

- ツリー作成部

ツリー作成部では，入力として XML 解析部で得られた XHTML のタグ情報と，パターン作成 ·解析部で得られたスクリプトのタグ情報を受け取る．そして，それぞれのタグ情報を統合して，要素の親子関係を表したツリーを作成する．

- 検証部

  検証部では入力としてツリー作成部で得られたツリーおよび DTD 解析部から得られた要素定義を受け取る．そして，そのツリーをルート要素から辿りながら，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 違反としてユーザーに警告を通知し，誤りがなければ正しい文書であることをユーザーに通知する．

4.3 ツールの実行例

図 14 のサンプルコードに対して，ECMAX を用いて検証した実行画面を図 19 に示す．

画面の構成を説明する．画面上部から，タイトルバー，メニューバー，コード表示ウィンドウ，メッセージ表示ウィンドウという構成になっている．

- タイトルバー
  タイトル「ECMAX」を表示する．他にも，最大化，最小化，終了することもできる．
- メニューバー
  メニューを表示する．メニューには，ファイルメニューとオプションメニューがある．
  - ファイルメニュー
    - Open メニュー
      ファイルを開く．メニューを実行すると，ファイルを選択するダイアログ (図 20) が現れる．
    - Start メニュー
      検証を開始する．入力ファイルが未選択だと，警告ダイアログ (図 21) が現れる．
    - Exit メニュー
      ECMAX を終了する．
  - オプションメニュー
    - DTD 選択メニュー (図 22)

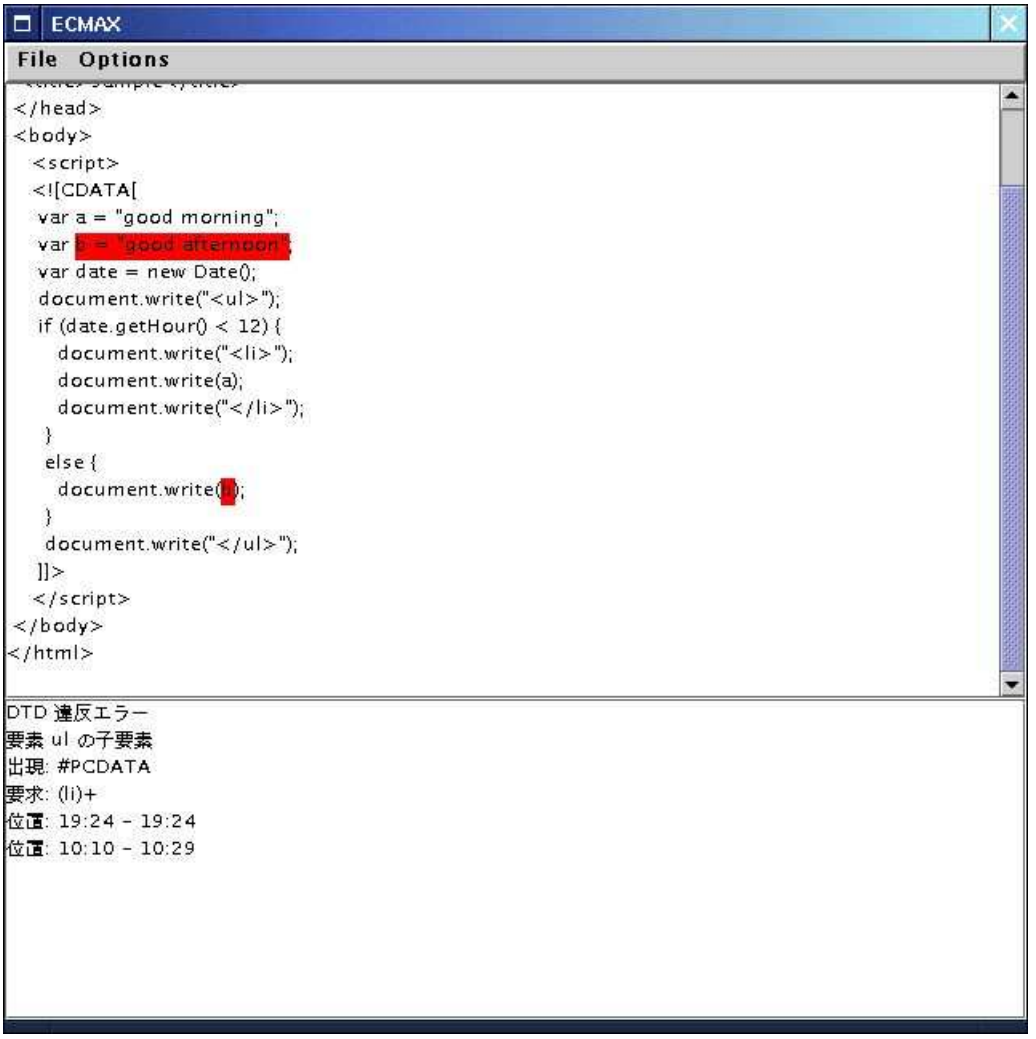

図 19: ECMAX の実行画面

- XHTML1-Strict

  DTD として XHTML1-Strict.dtd を使用する.

  デフォルトではこれが選択されている.

- XHTML1-Transitional

  DTD として XHTML1-Transitional.dtd を使用する.

- XHTML1-Frameset

  DTD として XHTML1-Frameset.dtd を使用する.

* フォントサイズ選択メニュー (図 23)

  - 12pt

フォントサイズを 12pt にする．
デフォルトではこれが選択されている．

- 14pt
  フォントサイズを 14pt にする．
- 16pt
  フォントサイズを 14pt にする．

- コード表示ウィンドウ
  入力ファイルのコードを表示する．また，エラーが起きた場合は，エラーが発生した場所，およびデータフロー解析結果から，エラーを起こす実行経路 (危険なパス) をハイライト表示してユーザーに提示することができる．(図 24)

- メッセージ表示ウィンドウ
  エラーメッセージなどを表示する．

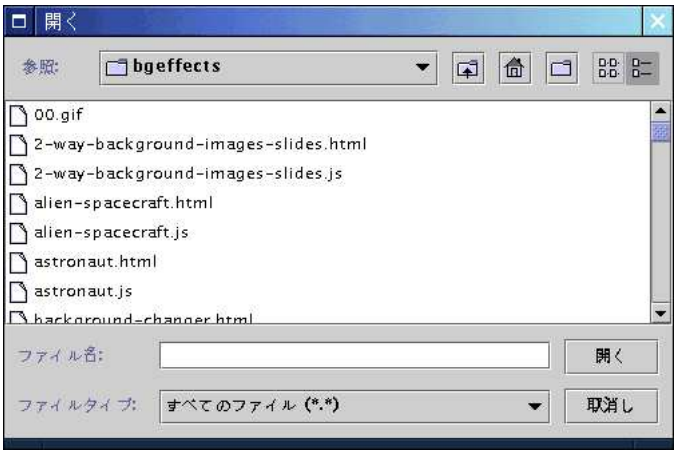

図 20: ファイル選択ダイアログ

図 21: ファイル未選択警告

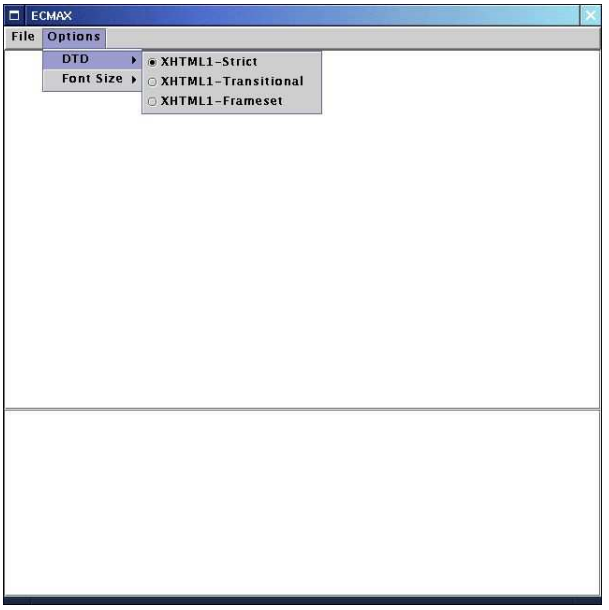

図 22: DTD 選択メニュー

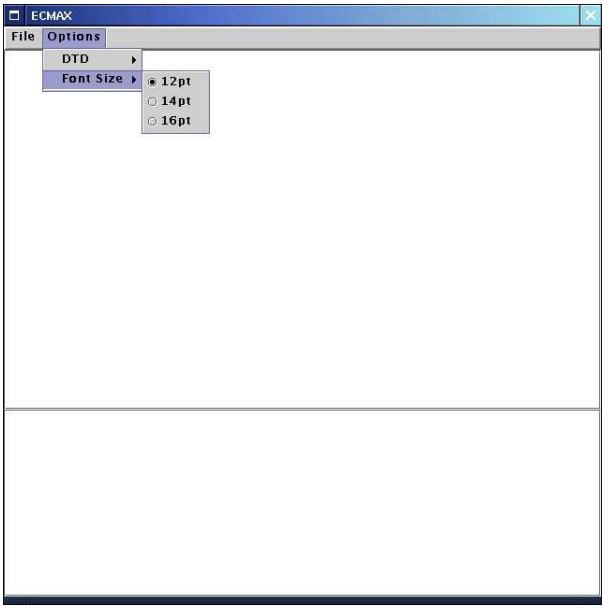

図 23: フォントサイズ選択メニュー

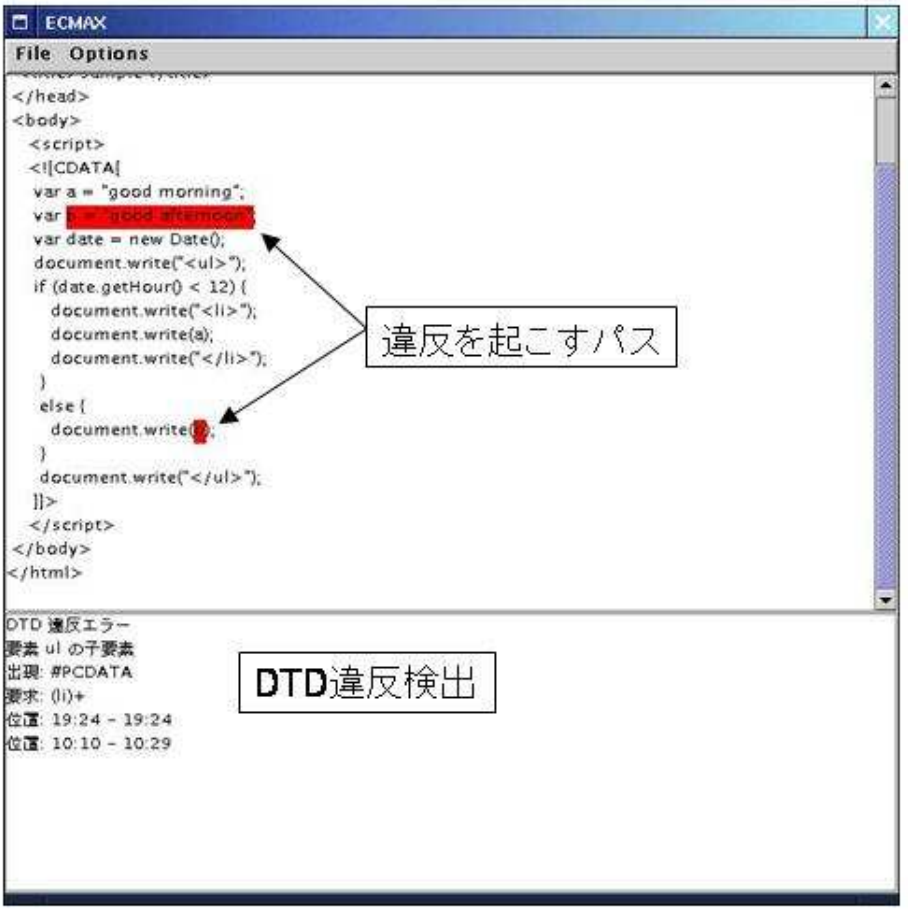

図 24: エラー (危険なパス) の表示

5　評価実験

本章では，インラインスクリプトとして ECMAScript で記述されたスクリプトを内部に含む XHTML 文書に対して，その構文を検証し，検証結果から本手法の評価を行なった．

5.1　実験対象

実験対象となるデータとして，実際にインターネット上で公開されている文書を収集した．それは，THE JAVASCRIPT SOURCE で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を対象とし，232 個のサンプルデータを収集した．その他の文書は，ウィンドウの制御など，文書を生成することとは別の目的のスクリプトを使用していたので，今回の実験には用いなかった．

＜実験データ＞

- JavaScript を含む HTML 文書: 232 個
  出展: THE JAVASCRIPT SOURCE
  http://javascript.internet.com

実験環境は以下の通りである．

- CPU: Pentium4 2GHz
- RAM: 2GB
- OS: FreeBSD 4.7
- JDK 1.3.1

5.2　実験方法

サンプルデータに対して，tidy という XHTML 変換ツールを用いて HTML から XHTML へ変換した．こうして得られた 231 個のインラインスクリプトを含む XHTML 文書について以下の実験を行なった．

- 実験 1 : 構文チェック
- 実験 2 : 厳密な DTD を用いて検証
- 実験 3 : 寛大な DTD を用いて検証

- 実験 4 : 既存ツールでの検証

＜実験 1 ＞
すべてのサンプルデータに対して，XHTML(XML) 構文チェックおよび ECMAScript 構文チェックを行なった．その上で，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なった．

＜実験 2 ＞
実験 1 を終了した時点で，構文に誤りがなかったサンプルデータに対して，DTD に XHTML1-Strict DTD を用いて検証を行なった．この DTD は，厳密にタグ付けが定義されている．

＜実験 3 ＞
実験 2 同様，実験 1 を終了した時点で，構文に誤りがなかったサンプルデータに対して，DTD に XHTML1-Transitional DTD を用いて検証を行なった．この DTD は，タグ付けに寛大であり，多くの XHTML 文書がこの定義を使用している．

これら 2 つの DTD の他にも，フレームを用いた XHTML 文書のための，XHTML1-Frameset DTD が存在するが，今回は，フレームを用いたデータがなかったため使用しなかった．

＜実験 4 ＞
既存の XHTML 検証ツール「htmllint」を用いて，以上の実験と同様の実験を行なった．

5.3 実験結果

＜実験 1 ＞
実験結果を表 3 に示す．

表 3: 実験 1 結果

XHTML 構文違反	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)
構文違反なし	113
合計	232

XHTML 構文エラーは，元のソースファイルの記述が間違っているものが多かった．

スクリプト構文エラーは，元のソースファイルの記述が間違っているものの他に，tidy が変換に失敗しているものもあった．

スクリプト内 XHTML 構文違反およびスクリプト内タグを含むタグ付け違反は，XHTML 変換ツールである tidy がスクリプト内の HTML タグを XHTML タグに変換するのに失敗している例が目立った．

＜実験 2＞

実験結果を表 4 に示す．

表 4: 実験 2 結果

DTD 違反	113 (35)
違反なし	0 (0)
合計	113 (35)

括弧内の数値は，不定値をテキストとみなして処理を行なった，条件付き検証数である．

すべての結果が DTD 違反であるという結果が出た．これは，XHTML1-Strict DTD が非常に厳密な定義を記述しているためである．一例を挙げると，body 要素の直接の子要素として，テキスト要素を記述してはいけないという定義や，一般的に用いられている center 要素，font 要素などが使用禁止という定義を行なっているため，メッセージを表示するという一般的なスクリプトなどはこの定義に違反してしまうのである．

＜実験 3＞

実験結果を表 5 に示す．

表 5: 実験 3 結果

DTD 違反	65 (12)
違反なし	48 (23)
合計	113 (35)

括弧内の数値は，不定値をテキストとみなして処理を行なった，条件付き検証数である．

XHTML1-Strinct DTD と比較して，より寛大な定義を記述しているため，DTD 違反は少なくなった．

＜実験 4＞
実験結果を表 6 に示す．

表 6: 実験 4 結果

DTD	Strict	Transitional
XHTML 構文違反	10	
DTD 違反	153	8
違反なし	69	214
合計	232	

多くの文書が違反なしという結果になった．

以上の結果をまとめると，表 7 のようになる．

表 7: 検証結果

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	

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 8 の通りである．

5.4 考察

検証結果から，既存の検証ツールでは検出できなかった，多くの構文違反や DTD 違反が見つかった．構文違反については，XHTML 変換ツールの不具合によるものも見受けられた

表 8: htmllint で違反が検出されなかった文書の ECMAX での検証結果

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
DTD 違反	34	63
違反なし	0	47
合計	69	214

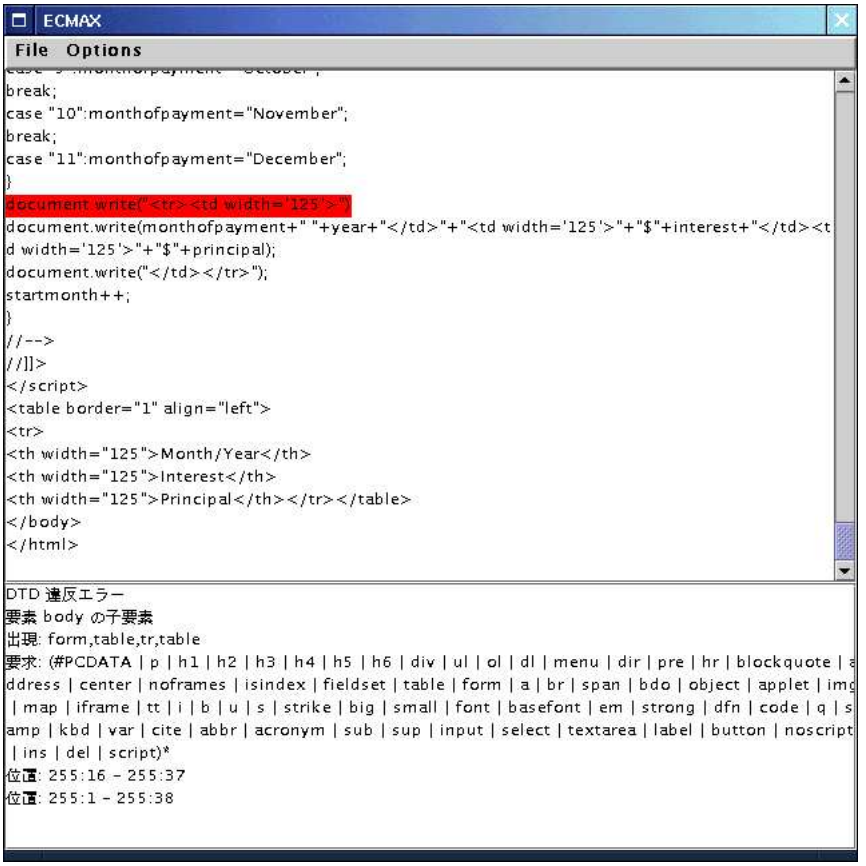

図 25: DTD 違反

が，もとのサンプルデータのエラーによるものもあった．

発見された DTD 違反の一例を見てみる．図 25 は，あるサンプルデータを検証した実行画面である．

このスクリプトは，<table> タグを記述し忘れ，その子要素である，<tr> タグを記述しようとして，DTD 違反が起こっている．このように，長いスクリプトを記述すると，タグの入れ子関係を正しく理解できずに，誤った XHTML 記述をしてしまうことが多い．そういった誤りを修正する上でも，本システムは有益であると言える．

また，不定値に関する検証について見てみると，およそ 3 割程度のサンプルデータが不定値を含むものとなっており，全体から見て小数である．さらに，その多くは，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スクリプトが多い．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ものであると思われる．

6 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なされている．

Claus Brabrand らは <bigwig> [6, 7, 8] 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．

Martin Kempa, Volker Linnemann は DOM [9] 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 [11] と，V-DOM 用の XML 処理言語である P-XML[10] を開発した．

Erik Mejier, Mark Shields は XML 処理用の関数型言語 XMλ [12] を開発した．また，細谷晴夫，Benjamin Pierce は XML 処理のための型付き関数型言語 XDuce [13, 14, 15] 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広く流通している既存の言語ではなく，独自の言語を対象としており，実用的とは言えない．

7 まとめ

本研究では，ECMAScript で記述されたスクリプトを内部に含む XHTML 文書を対象として，DTD の要素定義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．本手法では，スクリプトの出力文に対してデータフロー解析を行ない，出力文字列をパターンで表し，そのパターンを検証することで，構文の検証を行なった．そして，本手法に元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スクリプトを含む XHTML 文書に対して検証を行ない，本手法の評価を行なった．その結果，文書の構文間違い等を発見することができた．

また，将来の課題として以下のような点がある．

- 関数やライブラリへの対応
- エイリアスの解析
- XML Schema[19, 20, 22] への対応

謝辞

本論文を作成するにあたり，常に適切な御指導を賜わりました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 井上 克郎教授に心より深く感謝致します．

本論文の作成において，適切な御指導および御助言を頂きました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 楠本 真二 助教授に深く感謝致します．

本論文の作成において，適切な御指導および御助言を頂きました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 松下 誠 助手に深く感謝致します．

本研究を通して，適切な御助言を頂きました科学技術振興事業団 計算機科学技術研究員 山本 哲男 様に深く感謝致します．

最後に，その他様々な御指導，御助言等を頂いた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井上研究室の皆様に深く感謝致します．

参考文献

[1] W3 Consortium, HTML 4.01 Specification, http://www.w3.org/TR/1999/REC-html401-19991224, 24 December 1999

[2] Netscape Communications Corporation, JavaScript 1.1 Language Specification, http://www.netscape.com/eng/javascript/index.html, 1997

[3] W3 Consortium, Extensible Markup Language (XML) 1.0 (Second Edition), http://www.w3.org/TR/1998/REC-xml-20001006/, 6 October 2000.

[4] W3 Consortium, XHTML 1.0: The Extensible HyperText Markup Language (Second Edition), A Reformulation of HTML 4 in XML 1.0, http://www.w3.org/TR/2002/REC-xhtml1-20020801, 1 August 2002

[5] ECMA - Standardizing Information and Communication Systems, Standard ECMA-262, ECMAScript Language Specification, 3rd edition, http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM, December 1999.

[6] Claus Brabrand, Anders Moeller, and Michael I. Schwartzbach. The <bigwig> project. Submitted for publication. Available form http://www.brics.dk/bigwig .

[7] Claus Brabrand, Anders Moeller, and Michael I. Schwartzbach. Static Validation of dynamically generated HTML, SIGPLAN Notices, Supplement issue, pp.38-45, 2001.

[8] Claus Brabrand, Anders Mller, Anders Sandholm, and Michael I. Schwartzbach. <bigwig> - a language for developing interactive Web services. In preparation, 1999. http://citeseer.nj.nec.com/brabrand99ltbigwiggt.html

[9] W3 Consortium, Document Object Model (DOM) Level 1 Specification, Version 1.0, http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, 1 October 1998.

[10] Martin Kempa and Volker Linnemann, V-DOM and P-XML - towards a valid programming of XML-based applications, Information and Software Technology, vol.44, Issue 4, pp.229-236, 31 March 2002.

[11] Martin Kempa. VDOM: Dokumentenmodell for XML-basierte World-Wide-Web-Anwendungen. In Proceedings GI-Workshop Internet-Datenbanken, Berlin, pages 47-56, 2000. (in German).

[12] Erik Meijer and Mark Shields, XMλ:A functional language for constructing and manipulating XML documents, Draft, 1999.

[13] Haruo Hosoya and Benjamin Pierce,“XDuce: A Typed XML Processing Language”, World Wide Web and Databases. Third International Workshop WebDB 2000, Lecture Notes in Computer Science vol.1997, pp.226-244, Dallas, Texas, May 2000.

[14] H. Hosoya and B. C. Pierce. XDuce: an XML processing language. Preliminary report, Dec. 1999. http://citeseer.nj.nec.com/hosoya99xduce.html

[15] H. Hosoya, J. Vouillon, and B. C. Pierce. Regular expression types for XML. http:// www.cis.upenn.edu/ hahosoya/xduce.html, 2000. http://citeseer.nj.nec.com/article/hosoya00regular.html

[16] E.Pelegri-Llopart, L.Cable, Java Server Pages Specification, Version 1.1, Java Software, Sun Microsystems, 30 November, 1999.

[17] S.S. Bakken, A. Aulbach, E. Schmid, J. Winstead, L.T. Wilson, R. Lerdorf, Z. Suraski, A. Zmievski, J .Ahto, in S.S Bakken, E. Schmid (Eds.), PHP Manual, PHP Documentation Gruop, August 2001.

[18] Microsoft, JScript Language Reference, http://msdn.microsoft.com/library/default.asp? url=/library/en-us/script56/html/js56jslrfjscriptlanguagereference.asp .

[19] D. Beech, S. Lavrence, M. Maloney, N. Mendelsohn, and H. Thompson. XML schema part 1: Structures. May 1999. http://www. w3. org/TR/xmlschema-1/.

[20] P. V. Biron and A. Malhotra. XML Schema part 2: Datatypes. Available at http://www.w3.org/TR/xmlschema-2/, 2001.

[21] David Flanagan, JavaScript The Definition Guide 4th Edition, O'REILLY, 2002.

[22] Didier Martin, Mark Birbeck, Michael Kay, Brian Loesgen, Jon Pinnock, Steven Livingstone, Peter Stark, Kevin Williams, Richard Anderson, Stephen Mohr, David Baliles, Bruce Peat, Nikola Ozu 著, 石川直太監修, 鷺谷好輝訳, プロフェッショナル XML, インプレス, 2001.

[23] 中山幹男, 奥井泰弘, 改定版標準 XML 完全攻略 (上, 下), 技術評論社, 2001.

[24] 阿部友計, 図解 最新テクノロジー XHTML, ナツメ社, 2001.

[25] 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”, 2002 年電子情報通信学会総合大会講演論文集, D-3-5, pp.31, March 2002