複合 XML 文書の NVDL による検証のための ストリーム処理アルゴリズム

宮下 尚[†] 村田 真[†]

近年，名前空間を利用して，XML の多様な語彙を組み合わせて XML 複合文書が記述されることが多くなっている．このような XML 複合文書の検証を行う技術として，NVDL（Namespace-based Validation Dispatching Language）が注目されている．NVDL は，XML 複合文書を分割して，適切なスキーマによって検証することを可能とする．しかし，NVDL では，ストリーム処理を行うことは簡単ではない．なぜなら，複数の分割結果を許しているうえに，各々の分割結果に対して個別に検証器を起動できるためである．本稿では，NVDL による検証を，ストリーム処理で行うことができるアルゴリズムを提案する．このアルゴリズムでは，プッシュダウンオートマトンを用いることで，メモリ使用量の点で効率の良い処理を可能としている．

A Stream Algorithm for the NVDL Validation of Compound XML Documents

Hisashi Miyashita[†] and Makoto Murata[†]

Recently, many XML compound documents are composed with multiple XML vocabularies by using XML namespace. NVDL (Namespace-based Validation and Dispatching Language) is paid attention to validate such XML compound documents, which enables us to componentize schemas by namespaces by dividing XML documents and validating each fragment with an appropriate schema. However, streaming validation by NVDL is not easy because NVDL can divide documents in many ways and invoke a validator for each of them. In this paper, we show a novel algorithm that can divide and validate incoming XML documents with stream processing. This algorithm is efficient at the memory consumption owing to the use of a push-down automaton.

1. はじめに

近年，XML[5] は，一般の文書，メッセージ表現，グラフィックス表現等，様々な分野において，情報を記述するために利用されるようになっている．たとえば，Web 上で文書を記述するための XHTML ☆1，メッセージを表現するための SOAP ☆2，ベクトルグラフィックスを記述するための SVG ☆3等の多くの XML による語彙が提案されている．

XML では，名前空間[4] という機構によって，同一の文書中に，こういった多様な語彙を組み合わせて記述することを可能としている．この機構を用いることで，各々の語彙に対して，一意性のある名前空間を割り当てることができる．そのため，名前の衝突が起こることなく 1 つの文書に複数の語彙を組み合わせることが可能となる．名前空間を用いて複数の語彙を組み合わせて記述された文書を XML 複合文書と呼ぶ．XML 複合文書は，W3C における CDF[7] や XHTML や XForms ☆4，また，OASIS における Open Document Format for Office Applications（ODF）☆5等でも利用されており，最近では大きな注目を集めている．しかし，XML 複合文書の検証を従来までのスキーマ言語で行うことは，難しいという問題点があった．

NVDL（Namespace-based Validation Dispatching Language）[2] は，XML 複合文書に対して，名前空間ごとにインスタンスを適切に分割して，その分割された単位で検証をする技術である．NVDL で指定

† 日本 IBM 東京基礎研究所
Tokyo Research Laboratory, IBM Japan

☆1 http://www.w3.org/TR/2001/REC-xhtml11-20010531
☆2 http://www.w3.org/TR/2003/REC-soap12-part0-20030624
☆3 http://www.w3.org/TR/2003/REC-SVG11-20030114/
☆4 http://www.w3.org/MarkUp/Forms/
☆5 http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office

するスクリプトは，名前空間に基づく文書の分割規則と，分割された文書に対してのスキーマから構成される．NVDL を用いることによって，各々のスキーマは，自分の定義する XML 語彙の妥当性検証に集中することができる．最近では，NVDL は，W3C における複合文書の WG（CDF WG）[7] で，スキーマ定義言語として採用が検討されている．

1.1 NVDL による検証のストリーム処理の難しさ

XML 文書の検証処理において，ストリーム処理ができることは重要である．ストリーム処理ができない場合には，はじめにメモリ上に文書全体を展開する必要があるため，最低でも文書サイズに比例したメモリを消費し，しかも，メモリ上に文書を展開するまで処理を開始することができない．これは，大規模な文書を検証する場合や，レイテンシが小さいことが重要となるアプリケーションサーバでの処理では大きな問題となる．

NVDL 標準では，参照モデルによって NVDL 検証の意味が定義されている．この参照モデルは，NVDL における検証の意味を明確に定義するための処理モデルであるため，結果が同一であるならば実装はどのような処理を行ってもかまわない．この参照モデルは，メモリ上に保持された木構造を操作するアルゴリズムで記述されているため，ストリームでは処理できないモデルとなっている．

この参照モデルと等価なストリーム処理可能なアルゴリズムを実現することは，容易なことではない．なぜなら，名前空間によって分割された各々の文書の断片に対して，複数の操作を同時に可能としているためである．すなわち，文書の断片に対して，以下の複雑な動作を複数同時に処理しなくてはならない．

- 文書の 1 断片に対して，新たに 1 つまたは複数の検証器を起動する．
- 文書の断片を，親要素や祖先要素の一部として，すでに起動された検証器によって検証する．

図 1 に，検証処理の 1 つの可能性をあげる．この処理では，各 XML の文書断片ごとに新たに検証器を起動し，さらに同時に，親要素の一部もしくは祖先要素の一部として検証を行うことを指示している．

まず，図の一番上にある 5 つの三角形は，各々が 1 つの名前空間のみによって構成されている XML の文書断片である．この検証例では，各々の三角形に対して，新たに検証器を起動し，さらに，親要素もしくは祖先要素の一部として検証することの両方を同時に指示している．結果として，この例における NVDL の検証処理では，図中の下側に枠で囲んで表記した 9 種

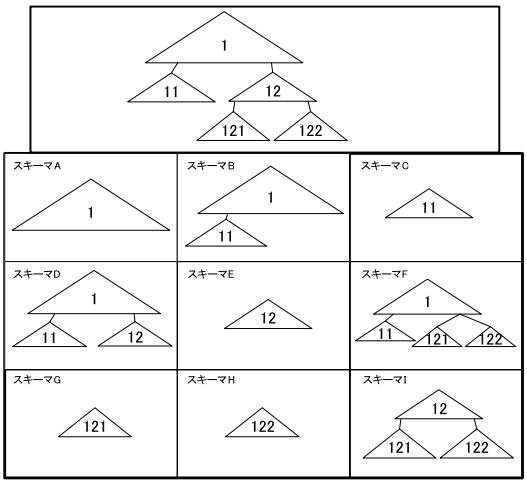


図 1 NVDL による検証処理の一例
Fig. 1 An example of validation processes by NVDL.

類の検証を同時に行わなくてはならない．この図は，文書断片 “1” だけからなる文書をスキーマ A に照らして検証する，文書断片 “1” および “11” からなる文書をスキーマ B に照らして検証する，文書断片 “11” だけからなる文書をスキーマ C に照らして検証する，文書断片 “1” と “11” と “12” からなる文書をスキーマ D に照らして検証する…ということを意味している．

このような検証をストリームで処理する場合には，たとえば，“121” の頂点にある要素がストリームに出現した場合には，スキーマ G によって単独で検証を開始し，文書断片 “1” にある要素の一部としてスキーマ F によって検証を行い，文書断片 “12” にある要素の一部としてスキーマ I によって検証を行う必要がある．特に，スキーマ F による検証では，“12” の文書断片を取り除き，“121” および “122” の文書断片を “1” の文書断片に接続して検証を行わなくてはならない．このような複雑な検証を，メモリ中に木構造を保持せずに行うことは単純なことではない．

NVDL では，このような複雑な分割および検証の処理を可能としているが，NVDL 標準では，ストリーム実装を実用上困難にしてしまう要件は巧妙に排除されて設計されている．たとえば，分割や再構成を行う処理は，木のルートからのパスだけで決定ができるように定められていて，大域的な情報が必要とならないように定められている．しかし，NVDL の検証をストリーム処理で行う効率的なアルゴリズムについての研究は，現在まで行われていなかった．

本稿では，以上のような複雑な NVDL の処理に対して，ストリーム実装ができる新しい検証アルゴリズムを提案する．このアルゴリズムでは，XML 文書の

分割と，分割された各々の文書の検証を同時に，しかもメモリ中に文書の全体を配置することなく行うことができる．提案するアルゴリズムでは，ストリーム中のイベントに対して，プッシュダウン・オートマトン（PDA）を適用して，分割および検証の規則を同時に複数割り当てることで処理を行う．この方式によるメモリ使用量は，文書インスタンスについて木構造の高さにのみ依存し，幅の大きさには依存しない．

本稿の以降の構成は，以下のとおりである．2 章で，NVDL について例を用いて説明する．3 章で，NVDL のストリーム実装可能なアルゴリズムを定義する．4 章では，NVDL の Java によるストリーム実装である SnRNV について説明し，5 章では関連研究について説明し，6 章で結論を述べる．

2. 例による NVDL の説明

本章では，例を利用して NVDL によってどのような処理ができるのか，また，そのような処理を行うためには，どのような記述をするのかについて説明する．簡単にいうと，NVDL とは，名前空間ごとに文書を分割し，記述された規則に基づいて分割された文書を再構成して検証を行う技術である．規則には，規則が適用される名前空間，規則が適用された文書に対する動作，その動作を適用後に，どの規則を次に適用するかを記述する．

2.1 スキーマをオープンにする NVDL

XML のスキーマにおいて，そのスキーマが対象としている名前空間に属する要素や属性だけを制約し，その他の名前空間（外部名前空間と呼ぶ）に属する要素や属性はすべて許すようなスキーマをオープンであるという．スキーマをオープンにすることは，拡張のために他の XML 語彙を許したいときには，特に重要な技法である．たとえば，ATOM☆標準におけるスキーマはオープンであるため，ATOM 文書中に XHTML を記述しても，ATOM スキーマに対してやはり妥当であるようにすることができる．

しかし，スキーマをオープンにして記述することは，既存のスキーマ言語，特に W3C XML Schema では難しい．なぜなら，このようなスキーマ言語では，ほとんどすべての内容モデルの定義に対して，外部名前空間の要素や属性を許すように記述する必要があるからである．

対照的に，NVDL では外部名前空間をすべて切り出して，その部分を許してしまうことによって，簡単に既存のスキーマをオープンにすることができる．

☆ http://www.ietf.org/rfc/rfc4287

```
<rules xmlns="...">
  <namespace ns="http://www.example.com/1">
    <validate schema="schema.xsd">
      <mode>
       <anyNamespace><allow/></anyNamespace>
      </mode>
    </validate>
  </namespace>
</rules>
```

図 2 スキーマ “schema.xsd” をオープンにする NVDL スキーマ．"..."は，NVDL 名前空間 URI である
Fig. 2 An NVDL Schema making a schema named “schema.xsd” open. "..." represents the NVDL namespace URI.

図 2 に，スキーマ schema.xsd をオープンにする NVDL スキーマの例をあげる．この NVDL スキーマは，まず，名前空間が http://www.example.com/1 である文書を schema.xsd を用いて検証するように，validate 要素で指示している．この validate とは指定されたスキーマで検証を行うことを指示するもので，NVDL では動作（action）と呼ばれる．この NVDL スキーマには，allow という動作も使われているが，これは，「あらゆる文書を許す」という意味である．動作には，他に attach や unwrap があり，これらの動作は分割された文書を組み替えるために用いられる．

この例において，http://www.example.com/1 名前空間の文書の検証中に，他の名前空間の要素が出現した場合には，validate 要素中の mode 要素の指示に従って検証を行う．mode 要素は，NVDL における状態であるモードを意味し，次の動作に遷移する際に，次にどの動作を選択するかを指定するために用いられる．ここでは，すべての名前空間（anyNamespace）について，あらゆる文書を許す（allow）ことが指示されているため，外部名前空間の部分は検証が行われないことになる．

たとえば，schema.xsd が，**図 3** に示したクローズドな（外部名前空間による要素を許さない）W3C XML Schema によるスキーマであるとする．この場合，そのまま schema.xsd を用いた場合には，**図 4** のような複合文書では検証に失敗するが，図 2 に示した NVDL を用いれば，このような外部名前空間を許した文書も妥当とすることができるようになる．

2.2 名前空間ごとにスキーマを切り替える NVDL

NVDL では，複数の名前空間に対してスキーマを切り替えることができる．このとき，複数のスキーマ言語を用いることもできる．たとえば，名前空間 A には

```
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified"
 targetNamespace="http://www.example.com/1">
 <xs:element name="People">
  <xs:complexType>
   <xs:sequence minOccurs="0"
                maxOccurs="unbounded">
    <xs:element name="Person"
                type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

図 3 クローズドな W3C XML Schema
Fig. 3 A closed W3C XML Schema.

```
<People xmlns="http://www.example.com/1"
xmlns:a="http://www.example.com/2">
 <Person>MIYASHITA, Hisashi</Person>
 <a:foo> anything you can write </a:foo>
</People>
```

図 4 外部名前空間を利用した XML 複合文書
Fig. 4 An XML compound document using an external namespace.

```
<rules xmlns="...">
  <namespace ns="D">
   <validate schema="D.rng">
    <mode>
     <namespace ns="A">
      <validate schema="A.rng"/>
     </namespace>
     <namespace ns="B">
      <validate schema="B.xsd"/>
     </namespace>
    </mode>
   </validate>
  </namespace>
</rules>
```

図 5 名前空間によってスキーマを切り替える NVDL スキーマ．"..."は，NVDL 名前空間 URI である
Fig. 5 An NVDL Schema swtching schemata by namespaces. "..." represents the NVDL namespace URI.

RELAX NG によるスキーマ `A.rng` を用い，名前空間 B では W3C XML Schema によるスキーマ `B.xsd` を用いるということが，簡単にできるようになっている．

図 5 に，名前空間によってスキーマを切り替える NVDL の例をあげる．この NVDL スキーマでは，名前空間 D に属する文書要素は `D.rng` で検証し，名前空間 A が出現した場合には `A.rng` で検証し，名前空間 B が出現した場合には `B.xsd` で検証することを指示している．

3. ストリーム実装可能な NVDL の検証アルゴリズム

本章では，NVDL のストリーム実装可能な検証用アルゴリズムを提案する．まず，初めに NVDL 標準の参照モデルについて概説し，次に，その参照モデルと同一の結果を得ることができる PDA によるアルゴリズムを提案する．最後に，分割された文書から検証器を選択して配送する機構について説明する．

3.1 NVDL 標準における検証の参照モデル

NVDL 標準の参照モデルにおける検証処理は，文書全体を同一の名前空間に属する文書の断片（セクションと呼ばれる）に分割することから始まる．次に，各々の要素セクションおよび属性セクションに対して動作を割り当てる．NVDL では，動作には `attach`, `unwrap` および `validate` の 3 種類が定義されている☆．最後に，割り当てられた動作の解釈を行う．セクションに割り当てられた動作が `attach` であれば，そのセクションを，文書中で祖先に位置していて，深さ方向から見て最も自分に近いセクションに取り付ける．動作が `unwrap` であれば，そのセクションを消去する．動作が `validate` であれば，そのセクションをルートとして検証を行う．

以下の項では，それぞれのステップについて順を追って説明する．

3.1.1 要素セクションおよび属性セクションへの分割

NVDL 標準の参照モデルにおける検証処理では，文書をセクションと呼ばれる単位に分割する．セクションは，要素セクション（element section）および属性セクション（attribute section）からなる．大まかにいうと，要素セクションとは，同じ名前空間に属する要素とその子孫からなる最大の文書断片である．そして，属性セクションとは，1 つの要素中の属性の集合から構成され，同じ名前空間に属する属性からなる最大の集合である．例として，**図 6** に示した XML 複合文書を考える．この文書を要素セクションと属性セクションに分割すると，**図 7** に示したように，3 個の要素セクションと，1 個の属性セクションが作られる．

要素セクション 1 の中にある *asn1* は，属性スロッ

☆ なお，NVDL では `deny` や `attachPlaceHolder` という動作も定義されているが，`deny` は `validate` 動作の一種と見なせ，`attachPlaceHolder` は `attach` の一種と見なせるため，本稿では割愛する．なお，2 章で紹介した `allow` 動作については，すべての文書を許すスキーマで検証せよと見なすことによって，`validate` 動作の一種と見なすことができる．

```
<ex xmlns="ns"
    xmlns:ns1="ns1" xmlns:ns2="ns2">
  <e1 ns2:b="b" ns2:c="c">t1</e1>
  <ns1:e2><e/></ns1:e2>
</ex>
```

図 6 XML 複合文書例
Fig. 6 An XML compound document.

要素セクション 1

```
<ex xmlns="ns"
    xmlns:ns1="ns1" xmlns:ns2="ns2">
  <e1 asn1 >t1</e1> esn2 </ex>
```

要素セクション 2

```
<ns1:e2 xmlns:ns1="ns1">esn3</ns1:e2>
```

要素セクション 3

```
<e xmlns="ns"/>
```

属性セクション 1

```
{ns2:b="b" ns2:c="c"}
```

図 7 要素セクション・属性セクションに分割された XML 複合文書
Fig. 7 An XML compound document divided into element and attribute sections.

トノード（attribute slot node）と呼び，もともとそこに属性セクション 1 があったことを指し示すための印である．同様に *esn2*，*esn3* は要素スロットノード（element slot node）と呼び，そこに要素セクション 2，3 がそれぞれあったことを示す．NVDL では，trigger を指定することによって，こうしてできあがった要素セクションをさらに分割することも可能であるものの，本稿では割愛する．

3.1.2 要素セクションおよび属性セクションに対する動作の割当て

NVDL によるスキーマの意味は，モードを状態と見なしたオートマトンによって定義することができる．このオートマトンを Mealy 機械として利用し，エッジに結び付けられた動作を，遷移先の状態に対応する要素セクションおよび属性セクションに対して適用することによって，各セクションに対して動作を割り当てることができる．

NVDL による Mealy 機械は形式的に以下のように定義される．

定義 1（NVDL による Mealy 機械の定義）
NVDL による Mealy 機械 M は五つ組 $M = (Q, \Sigma, q_0, \Delta, R)$ から構成される．Q はモードの集合である．$\Sigma = (\{\mathtt{as}, \mathtt{es}\} \times N)$ は，入力アルファベットで，属性セクション（as）か要素セクション（es）かを示すシンボル，および，名前空間 URI から構成される（ここで，名前空間 URI の集合を N で表した）．q_0 は開始モードである．$\Delta = \{\mathtt{validate}, \mathtt{attach}, \mathtt{unwrap}\}$ は動作の種類の集合である．R は $(Q \times \Sigma)$ から $(Q \times \Delta)$ への遷移の対応で，モードと遷移先の名前空間 URI から，次のモードおよび遷移先の要素もしくは属性セクションに割り当てる動作を決める． □

```
<rules xmlns="...">
 <namespace ns="ns">
  <validate schema="ns-schema.rng">
   <mode>
    <namespace ns="ns1">
     <unwrap><mode>
      <namespace ns="ns">
       <attach/>
      </namespace>
     </mode></unwrap>
    </namespace>
    <namespace ns="ns2" match="attributes">
     <validate schema="ns2-schema.rng"/>
    </namespace>
   </mode>
  </validate>
 </namespace>
</rules>
```

図 8 NVDL スキーマの例．... は，NVDL 名前空間 URI である
Fig. 8 An NVDL Schema. ... represents the NVDL namespace URI.

NVDL 標準では，こうして作られる Mealy 機械の全域性を保証するために，ディフォルトの anyNamespace ルールが自動的に割り当てられることになっているが，単純化のためにここでは省略する．なお，NVDL では 1 つの遷移に複数の validate 動作を割り当てることができるため，Mealy 機械は非決定的になりうる．つまり，1 つのセクションに複数の validate 動作が割り当てられる可能性がある☆．

この定義に基づいて，**図 8** の NVDL スキーマを Mealy 機械 $M = (Q, \Sigma, q_0, \Delta, R)$ で定義すると，$Q = \{q_0, q_1, q_2\}$，$\Sigma = \{\langle \mathtt{ns}, \mathtt{es}\rangle, \langle \mathtt{ns1}, \mathtt{es}\rangle, \langle \mathtt{ns2}, \mathtt{as}\rangle\}$ となる．そして，遷移関数 R は以下のように定義される．

$(q_0, \langle \mathtt{ns}, \mathtt{es}\rangle) \to (q_1, \mathtt{validate})$
$(q_1, \langle \mathtt{ns2}, \mathtt{as}\rangle) \to (q_1, \mathtt{validate})$
$(q_1, \langle \mathtt{ns1}, \mathtt{es}\rangle) \to (q_2, \mathtt{unwrap})$
$(q_2, \langle \mathtt{ns}, \mathtt{es}\rangle) \to (q_2, \mathtt{attach})$

これを模式的に図にすると，**図 9** のようになる．

次に，この Mealy 機械を用いて，図 7 で示した分割された文書に対して動作を割り当てると，**図 10** の

☆ 複数の attach もしくは unwrap 動作を指定することは禁止されている．

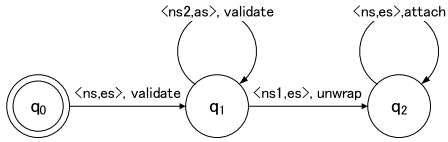


図 9 図 8 の NVDL スキーマに対応する Mealy 機械．各エッジには入力シンボル（名前空間 URI および，属性セクション（as）か要素セクション（es）かを表すシンボルのペア）と，適用される動作が記述されている

Fig. 9 A Mealy machine of the NVDL schema in Fig. 8. In each edge, we specify an input symbol (a pair of a namespace URI and a symbol specifying element section es or attribute section as) and applicable actions.

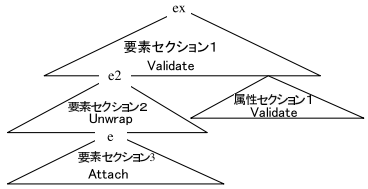


図 10 図 7 で示した XML 文書に対して，図 9 による Mealy 機械で動作を割り当てた結果．各々の三角形は要素もしくは属性セクションを表している．また，各セクションに割り当てられた動作が三角形中に書かれている

Fig. 10 The result obtained by applying the Mealy machine in Fig. 9 to the XML document in Fig. 7. Each element or attribute section is denoted by a triangle in which we also denote the assigned action.

ように模式的に表すことができる．

3.1.3 動作の解釈

属性セクションおよび要素セクションに動作が割り当てられた後は，それらのセクションから文書を再構成し，検証を行う．要素セクションに対する動作と，属性セクションに対する動作は，以下に示すように少々異なる．

要素セクションに validate が割り当てられている場合には，その要素セクションを文書要素として，validate 動作に割り当てられている検証器で検証を行う．要素セクションに attach が割り当てられている場合は，親の要素セクションの対応する要素スロットノードに取り付ける．要素セクションに unwrap が割り当てられている場合には，その要素セクションが持つすべての要素スロットノードに（attach によって）取り付けられたすべての要素セクションを，親の要素セクションの対応する要素スロットノードに取り付ける．

一方で，属性セクションに validate が割り当てられている場合には，その属性セクションに対して virtualElement という名前の仮の要素に対して属性

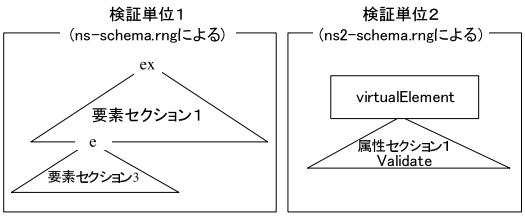


図 11 図 10 で示した動作を解釈することによって行われる検証処理

Fig. 11 Validations induced by the assigned actions described in Fig. 10.

セクションを取り付けて，指定された検証器で検証を行う．属性セクションに attach が割り当てられている場合には，親の要素セクションの対応する属性スロットノードに取り付ける．属性セクションに unwrap が割り当てられている場合には，何も行わない．

なお，1 つのセクションに，複数の validate 動作が割り当てられていた場合には，すべての validate 動作の解釈を行う．

以上の動作の解釈を図 10 で示した例に対して適用すると，**図 11** で示すように検証が行われる．すなわち，要素セクション 2 の部分が unwrap 動作によって削除され，かわりに要素セクション 3 が要素セクション 1 に attach 動作によって取り付けられ，これが検証単位 1 となる．そして，（図 8 での 3 行目の）validate 動作によって，ns-schema.rng を用いて検証されることになる．また，属性セクション 1 は，（図 8 での 12 行目の）validate 動作によって，仮要素である virtualElement に取り付けられて，ns2-schema.rng を用いて検証されることになる．

結果として，この例においては，ns-schema.rng を用いて

```
<ex xmlns="ns"
    xmlns:ns1="ns1" xmlns:ns2="ns2">
  <e1>t1</e1>
  <e/>
</ex>
```

が検証されることになる．また，ns2-schema.rng を用いて

```
<virtualElement xmlns="..."
 xmlns:ns2="ns2" ns2:b="b" ns2:c="c"/>
```

が検証されることになる（... には virtualElement のために予約されている名前空間 URI である http://purl.oclc.org/dsdl/nvdl/ns/instance/1.0 が入る）．

3.2 ストリーム処理可能なアルゴリズム

前節で説明した NVDL の参照モデルによるアルゴ

リズムによって，確かに文書を分割して検証することが可能である．しかし，参照モデルによって定義されている操作は，木構造の変形をともなうため，同一の操作をそのまま実装する場合には，すべての木構造をメモリ中に配置する必要がある．たとえば，図 7 における要素セクション 3 の要素 e の処理を行う場合を考えよう．このためには，1) 要素 e に対する要素セクションの特定，2) 要素セクションに対する動作の特定，3) 割り当てられた動作の解釈を行う必要がある．参照モデルで，これらの動作を決定するためには，要素セクション 2 に割り当てられた unwrap 動作，および，attach 動作のために要素セクション 1 の要素スロットノードの情報 *asn1* を必要とする．さらに複雑なことに，1 つのセクションには複数の動作が割り当てられるため，祖先のセクションの木構造がメモリ上にない場合に，この 2 つの情報を得るようなアルゴリズムを作ることは容易なことではない．

本節では，NVDL の参照モデルと同一の結果を得ることができる，ストリーム処理（SAX☆1 による処理）が可能なアルゴリズムを提案する．

このアルゴリズムでは，XML の階層構造を SAX ストリームで認識するために，スタックを導入する．すなわち，プッシュダウンオートマトン（PDA）を利用して，各 SAX イベントに対して動作を割り当てる．動作を割り当てた後，スタックにおける階層構造から，そのイベントが，どの検証単位に属するのかを決定する．

3.2.1 PDA による動作の割当て

3.1.2 項で見たとおり，属性・要素セクションを入力列と見なせば，動作の割当ては非決定有限オートマトン（NFA）で行うことができる．しかし，NFA では，SAX ストリームにおけるイベント列に対して入れ子の構造を認識することができない．なぜなら入れ子の階層は，有限でおさえられないため，正規言語では表現できないためである☆2．

そのため，本アルゴリズムでは，スタックを導入して入れ子の構造を認識しつつ，3.1.2 項をもとに，以下のような PDA を作成する．

定義 2（ストリーム処理のための PDA） ストリーム処理をするための PDA M' は六つ組 $M' = (Q', \Sigma', \Gamma', q'_0, v_0, R')$ から構成される．$Q' = Q \times (\Delta \cup \{\bot\})$ は，モードと動作（動作を指定しないこともできる．この場合は $\bot$ と表記する）のペアから構成される．Σ' は SAX イベントから自然に構成する．簡単化のために，ここでは，XML の開始タグおよび終了タグのみを考える．$\Gamma' = (Q' \times N)$ は，スタックアルファベットであり，状態と名前空間 URI のペアとする．q'_0 は初期状態であり，ここでは，$\langle q_0, \bot \rangle$ と定義する（q_0 に対応し，動作は指定しない）．$v_0 \in \Gamma'$ はスタックの初期状態で，ここでは，q'_0 と，どの名前空間とも衝突しない名前空間とのペアであるとする☆3．R' は $(Q' \times \Sigma' \times \Gamma')$ から $(Q \times \Gamma'^*)$ への遷移の関係である． □

定義 2 による PDA を作成するためには，まず，定義 1 による Mealy 機械 M を作成する．この M に対応する PDA M' は，以下のようにして作られる．

(1) すべての状態において，同一要素セクション中では，スタックに同一の名前空間を積み込むだけで遷移を行わないようにする．
(2) 開始タグが現れた場合には，スタックに以前の状態をプッシュし，R の指定に従って新しい状態に移行する☆4．
(3) 終了タグが現れた場合には，スタックをポップして以前の状態に戻る．

上記の手順 (1)–(3) を形式的に記述すると，それぞれ以下のようになる．

(1) Q' 中のすべての状態 q'，名前空間 URI n に属するすべての開始タグ s について，$(q', s, (q', n)) \to (q', (q', n)(q', n))$ が R' に含まれるようにする．
(2) $(q_1, \langle n, m \rangle) \to (q_2, d)$ が R に含まれているならば（n は名前空間 URI，m は as，es のいずれか，d は動作），Q' 中のすべての状態 q'，n とは等しくないすべての名前空間 URI n'，名前空間 URI n に属するすべての開始タグ s，Δ 中のすべての動作 d' に対して，$(\langle q_1, d' \rangle, s, (q', n')) \to (\langle q_2, d \rangle, (q', n')(\langle q_1, d' \rangle, n))$ が，R' に含まれているようにする．
(3) Q' 中のすべての状態 q'_1，q'_2，すべての名前空間 URI n，すべての終了タグ e に対して，$(q'_1, e, (q'_2, n)) \to (q'_2, \emptyset)$ が R' に含まれているようにする．

☆1 http://www.saxproject.org/

☆2 X を開始タグ，$\bar{X}$ を終了タグとすると，$X^n \bar{X}^n$ の構造が現れる．しかし，pumping lemma より，これは正規言語ではないことが分かる．

☆3 これは，最初のルート要素を，必ずセクションと認識させるためのダミーのスタック要素である．

☆4 なお，スタックに動作を記憶するためには，状態に動作を含める必要がある（Q' の定義を参照せよ）．このためには，Mealy 機械を Moore 機械に変換する必要があるので，手順 (2) ではそれもあわせて行う．

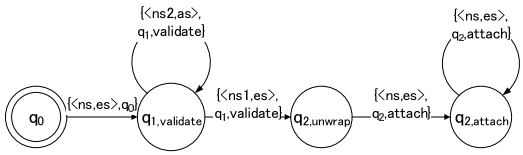


図 12 図 9 の Mealy 機械に対応する PDA．各状態には，図 9 の Mealy 機械の状態と，遷移時に適用される動作のペアが記述されている．各エッジには入力シンボルと，スタックにプッシュされる遷移前の状態が記述されている

Fig. 12 An PDA corresponding to the Mealy machine in Fig. 9. In each state, we specify a pair of a state of the Mealy machine and an action applied when the state is selected. In each edge, we specify an input symbol and the previous state pushed to the stack.

なお，この PDA の構成では，属性セクションの処理については簡単化のために省略した．属性セクションについては，同一名前空間に属する属性で，属性の順序を並べ替えることによって，同様に PDA で処理をすることができる．

図 12 に，図 9 の Mealy 機械に対応する簡略化して表記した PDA を示す．この図においては，単純化のために，上記 (1)，(2) に対応する遷移は記述されていない．(1) については，すべての状態に対して，自分自身の状態に対応する遷移を追加することとなる．そして，(2) については，自分および他のすべての状態に対する遷移を追加することとなる．

このようにして構成された PDA を用いれば，各イベントに割り振られた状態 $\langle q, d\rangle$ から，動作 d を決定することができる．この PDA は，もともとの Mealy 機械が非決定的なので，非決定的となる．したがって，1 つのイベントには複数の状態が割り当てられる可能性があるため，動作についても複数が割り当てられる可能性がある．

3.2.2 ストリーム処理における動作の解釈

前項の手順で，SAX イベントに動作を割り当てることができる．つまり，各イベントごとに，図 10 に示したものと同じように，動作を割り当てられたことになる．この項では，各イベントに割り当てられた動作から検証器を選び，適切に配送を行う方法の説明を行う．以下の説明においては，図 10，図 11 が参考となる．

対象の SAX イベントが `startElement` である場合（開始タグに対応する），まず属性から処理を行う．この場合，前項で説明したとおり，同一名前空間の属性はまとめて処理する必要がある．属性に割り当てられた動作が `validate` の場合には，仮の要素である `virtualElement` 要素に同一の名前空間の属性を取り付けて，指定された検証器で検証を行う．図 6 の例では，要素 `e1` に取り付けられている属性には，`validate` 動作が割り当てられるため，`ns2:b` および `ns2:c` 属性については，図 11 中の右側で示したように `virtualElement` 要素に取り付けて検証を行う．割り当てられた動作が `attach` の場合には，もともとの要素にその属性を残す．そして，後述する要素に関する処理に従って検証器に配送される．これによって，もともとの属性スロットノードに取り付けることと同じ効果を得ることができる．割り当てられた動作が `unwrap` の場合には，その属性については，もともとの要素から取り除いて，何も行わない．

引き続き，要素に関する処理を行う．割り当てられた動作が `validate` である場合で，その同一要素セクションについて，まだ検証器が起動されていない（まだ新しい検証単位の文書が開始されていない）のであれば，`validate` 動作に対応した検証器に対して新しい検証単位を開始して，続いて，対象の SAX イベントをその検証器に送る．図 6 の例では，要素 `ex` で，最初に `validate` 動作が割り当てられるため，新しく検証器を起動して，要素 `ex` についての SAX イベントを送出する．もし，すでに検証器が起動されている場合には，その検証器に対して，引き続き対象となる SAX イベントを送る．

割り当てられた動作が `attach` の場合には，PDA 中のスタックをさかのぼり，動作が `validate` であるところに対応した検証器に対して，対象の SAX イベントを送出する（対象の SAX イベントは，`validate` 動作が割り当てられた要素の子要素なので，すでに検証単位は開始されているはずである）．これによって，`validate` 対象の要素セクションに該当する要素スロットノードに取り付けたことと同じ効果を得ることができる．図 6 の例では，要素 `e` で `attach` 動作が割り当てられることになる．この場合には，スタックをさかのぼると，まず，要素 `e2` で割り当てられた `unwrap` 動作が見つかる．しかし，これは動作が `validate` ではないため，さらにスタックをさかのぼる．次に，要素 `ex` で割り当てられた `validate` 動作が見つかる．そこで，この `validate` 動作に対応する検証器に対して，要素 `e` に関する SAX イベントを送出することになる．

割り当てられた動作が `unwrap` である場合には，その SAX イベントについては何もしない．図 6 の例では，要素 `e2` についての SAX イベントは，検証器に送られない．

以上のように処理を行うことによって，図 6 の例で

は，図 11 の左側で示したように，要素セクションについての検証が行われることになる．

3.1.3 項で説明したように，やはり，ここでも 1 つのイベントに複数の `validate` 動作が割り当てられることが起こりうる．この場合，すべての解釈について検証を行う必要がある．このとき，各々の解釈についてスタックの状態が異なることに注意する必要がある．

以上の手順によって，NVDL の検証をストリームで処理することができる．

3.2.3 ストリーム処理アルゴリズムの計算量

ストリーム処理アルゴリズムの計算量は，以下の 2 点に分類される．

(1) NVDL スキーマ定義から生成される PDA の計算量

(2) 生成された PDA が実行時に必要する計算量

まず (1) について考察する．PDA の状態は，たかだか NVDL のスキーマにおけるモードの数に比例する回数で生成可能である．また，PDA の遷移の構築にかかるステップは，たかだかその 2 乗程度でおさえられる．したがって，NVDL スキーマのサイズに対する PDA 構築の計算量は，PTIME となる．

次に (2) の実行時の計算量について考察する．NVDL の検証動作において，要素セクションの階層が 1 段深くなるたびに，1 回の `validate` 動作で，2 個の検証器を新たに起動する場合を考える．この場合，要素セクションの階層構造の数 n に対して，$O(2^n)$ の検証器を起動することとなる．すなわち，最悪で EXPSPACE の計算量が必要となることを意味している．このため，入力インスタンスの木構造の高さに対して指数の空間計算量を持つことになってしまう．しかし，木構造の幅については，スタックを消費することはないため，メモリ量は有限となる．また，遷移が複数に分かれることもないので，入力に対して比例したステップしか必要としない．

4. 実　　装

我々は，3.2 節で示したアルゴリズムをもとに，NVDL 検証器の Java による実装 SnRNV を作成した．SnRNV は，SAX ストリームによる入力を NVDL の指定に基づいて分割し，JAXP☆に対応した検証器を呼び出すことができる．**図 13** に，SnRNV の構成図を示す．SnRNV は，入力として NVDL スキーマと検証対象となる XML 複合文書を受け取る．NVDL スキーマ中では，他の W3C XML Schema や RELAX NG 等によるスキーマが参照されており，必要となった時点で SnRNV は適切にそれらのスキーマを読み込み，JAXP を用いて検証器として起動する．入力文書は，SAX パーサによって読み込まれ，NVDL スキーマの指示に従って選択された検証器に，SAX イベントとして配送されることになる．

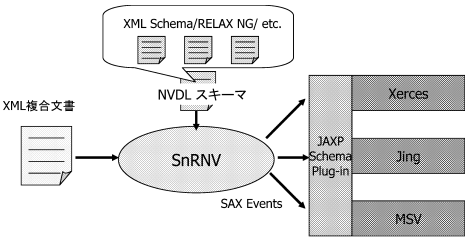


図 13 SnRNV の構成図
Fig. 13 SnRNV's block diagram.

NVDL の SAX 実装において，特に問題になることが名前空間宣言の取扱いである．SAX 標準によると，名前空間が有効になる前に `startPrefixMapping` メソッドを有効になる名前空間ごとに呼び出さなければならず，やはり，無効になる前に `endPrefixMapping` メソッドを呼び出さなくてはならない．ところが，文書をそのまま分割して検証器を呼び出してしまうと，名前空間宣言が失われてしまい，正常に `startPrefixMapping` や `endPrefixMapping` が呼ばれないことが起こりうる．このため，新しく文書を分割して検証器に送り出す際に，現在有効な名前空間について `startPrefixMapping` を呼び出し，分割された文書の終了時には `endPrefixMapping` を開始した際に有効にした名前空間について呼び出す必要がある．

4.1 メモリ使用量

NVDL 標準[2] の付録 D.3 にある XHTML と XForms の複合文書の NVDL の例について，SnRNV を適用し，メモリ使用量を計測した．本アルゴリズムが使用するメモリ量は，理論的には PDA の状態の数，および，PDA のスタックの最大消費量にのみ依存する．PDA の状態の数は NVDL のスキーマにより一意に決定されるため，文書量には依存しない．スタックの最大消費量は，入力文書の木構造の最大の高さによって決定され，高さが一定であるならばスタックの最大消費量も一定のはずである．したがって，本実験では，木構造の高さおよび文書サイズの最大値は一定でありながらも，文書サイズが変化するようにして実験を行った．

実験に用いた複合文書は，XHTML と XForms を組み合わせたものである．**図 14** に実験に用いた XML 複

☆ `http://java.sun.com/webservices/jaxp/index.jsp`

```
<html
 xmlns:xforms="http://www.w3.org/2002/xforms"
 xmlns="http://www.w3.org/2002/06/xhtml2">
  <head><title>title</title></head>
  <body>
    <table>
      <xforms:repeat
        id="lineset"
        nodeset="/my:lines/my:line">
        <tr><td>
          <xforms:input ref="my:price">
   <p><xforms:label>Item</xforms:label></p>
          </xforms:input>
          <!-- (A) -->
        </td></tr>
      </xforms:repeat>
    </table>
    <!-- (B) -->
  </body>
</html>
```

図 14 実験に用いた XHTML および XForms からなる XML 複合文書
Fig. 14 An XML compound document of XHTML and XForms used in the experiment.

合文書を示す．図中の `<!-- (A) -->` において，`table` 要素をネストさせることによって，木構造の高さを変え，図中の `<!-- (B) -->` で，`table` 要素を繰り返すことによって文書サイズを変化させることができる．`<!-- (B) -->` の繰返し方法は，木構造の高さを決められた最大から，最小の高さ（10 段）まで順に変化させ，最小の高さまでたどり着いたら，また決められた最大の高さに戻すことで変化させた．NVDL スキーマは，NVDL 標準中の D.3 節における XHTML 2.0 と XForms の複合文書のためのスキーマをそのまま用いた．

環境は，Sun JDK のバージョン 1.5.0_06，Windows XP SP2，Thinkpad X41（CPU Intel Pentium M 1.60 GHz）を用いた．RELAX NG 検証器には Multi Schema Validator（Version 20030108）（MSV）を利用した．

実験によって計測されたメモリ消費量を**図 15** に示す．グラフが示すとおり，メモリ使用量は入力インスタンスにおける木構造の最大の高さのみに依存し，幅には依存しないことが分かる☆．通常の XML 文書では，ネスト構造が極端に深い場合には，可読性が落ちるばかりでなく，処理するプログラムも複雑になりやすくなるため，あまりにも高い木構造は避けることが

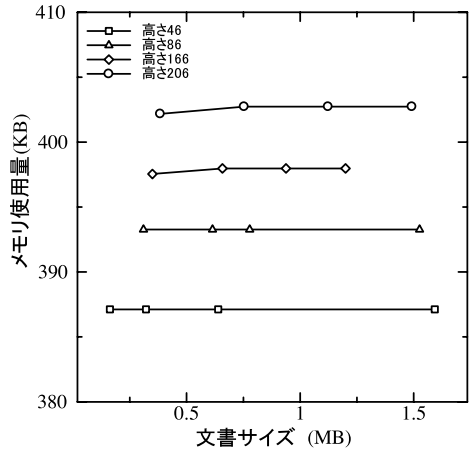


図 15 SnRNV におけるメモリ消費量
Fig. 15 Memory consumption by SnRNV.

一般的である．したがって，本アルゴリズムにおいて，木構造の高さが増えることで消費メモリ量が増えることは実用上問題とはならない．

この結果は，本アルゴリズムによって，理論どおり，文書サイズによらず一定のメモリのみを消費することが可能であること，すなわち，実際にストリーム処理で NVDL の検証が可能になることを示している．

5. 関連研究

NVDL のように，XML 文書を分割して検証を行うという手法について，それほど多くの研究が行われていたわけではない．

文書を名前空間単位で分割して検証を行うというアイデアは，RELAX 名前空間[1] までさかのぼることができる．RELAX 名前空間では，名前空間ごとに検証単位を指定することができる．しかし，現在の NVDL にあるようなモードの概念はないため，要素セクションや属性セクションの階層構造を認識することはできなかった．また，`attach`，`unwrap` のように構造を組み替えて検証することもできなかった．

その後，Clark は Modular Namespaces（MNS）☆☆ を提案した．MNS では，モードの概念が導入され，セクションの階層構造によって検証動作を変更できるようになっていた．後に Clark は NVDL の原型である Namespace Routing Language（NRL）☆☆☆ を提案した．NRL には構造組換えの機能もあり，NVDL とほとんど同等の機能を備えていた．

なお，NVDL は，NRL からの知見をもとに

☆ メモリ使用量にわずかに変動が観測される．これは Garbage Collection のタイミングによる計測誤差と推察される．

☆☆ http://www.thaiopensource.com/ relaxng/mns.html
☆☆☆ http://www.thaiopensource.com/ relaxng/nrl.html

ISO/IEC で標準化されたものである．

6. おわりに

本稿では，XML 複合文書を検証する手段としての NVDL について説明し，ストリーム実装可能なアルゴリズムを示した．また，実際に Java による実装である SnRNV を作成し，XHTML および XForms による複合文書を検証した際には，メモリ使用量は文書の木構造の高さのみに依存し，木構造の幅には依存しなかった．

NVDL は，XML 複合文書を適切に分割することによって，すべての XML 語彙についての知識がなくても，個々の名前空間の処理を統合することによって，文書全体の検証処理が可能であるように設計されている．

XML 複合文書は，今後利用の拡大が見込まれる分野であり，今後は，検証処理以外にも利用可能な，より汎用性のある配送器（Dispatcher）として NVDL を活用できるようにしたい．そのためには，XML 複合文書の処理自身の研究，特にソフトウェアコンポーネントに分割された文書をどのように配信すればよいのかを，考慮する必要があると考えている．

参考文献

1) XML 正規言語記述 RELAX 名前空間 (2003). 標準情報 (TR) TR X 0044:2001.
2) Information technology — Document Schema Definition Languages (DSDL) — Part 4: Namespace-based Validation Dispatching Language (NVDL) (2006). ISO/IEC 19757-4.
3) Berstel, J. and Boasson, L.: XML Grammars, *MFCS*, pp.182–191 (2000).
4) Bray, T., Hollander, D. and Layman, A.: Namespaces in XML, W3C Recommendation (Jan. 1999). http://www.w3.org/TR/1999/REC-xml-names-19990114
5) Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E. and Yergeau, F.: Extensible Markup Language (XML) 1.0 (3rd Edition), W3C Recommendation (Feb.2004). http://www.w3.org/TR/REC-xml-20040204
6) Fülöp, Z. and Vogler, H.: *Syntax-Directed Semantics*, Monographs in Theoretical Computer Science, An EATCS Series, Springer (1998).
7) Mehrvarz, T., Appelquist, D., Pajunen, L. and Quint, J.: Compound Document by Reference Framework 1.0, Editor's Draft (June 2006). http://www.w3.org/2004/CDF/specs/CDR/wp-1/cdf.html

付録

A.1 参照モデルにおける Mealy 機械と本稿における PDA による Moore 機械の等価性

この章では，3.1 節における参照モデルによる Mealy 機械と 3.2 節における PDA による Moore 機械との等価性の証明の概略を与える．この 2 つの機械には，前者は，名前空間による tree を入力として持つのに対し，後者は，木構造構築前の word を入力として持つという大きな違いがある．したがって，準備として，この 2 つの違いに対処できるように形式化する必要がある．ポイントは，PDA の入力を一般の word とするのではなく，XML イベントの入力である Dyck word[3] を用いて定式化することである．これによって非整形式の XML イベントは入力として取り扱わないことになるが，SAX では整形式の XML であることをパース時にチェックするため，これが問題になることはない．

以降，シンボル Σ による unranked tree を T_Σ と表記する．

はじめに，参照モデルによる Mealy 機械の変換を形式的に定義する．

定義 3（Mealy 機械による変換） 定義 1 による Mealy 機械 $M = (Q, \Sigma, q_0, \Delta, R)$ による変換 $t = M(q(s))$ は $s \in T_\Sigma$ から，ラベル付けされた木の集合 $t \in 2^{T_{\Sigma\times\Delta}}$ への関数として，以下のように帰納的に定義される．

- $M(q(s)) = M^\omega_\perp(q(s))$ とする．ただし $\omega \in \Sigma$ は，名前空間が他のどの名前空間とも異なる仮想のシンボルとする．
- $s = \sigma(s_1, \ldots, s_n)$ について，

$$M^{\sigma'}_d(q(s)) = \begin{cases} \{\langle \sigma, d\rangle(M^\sigma_{q,d}(s_1, \ldots, s_n))\} \\ \qquad (\mathrm{NS}(\sigma) = \mathrm{NS}(\sigma')) \\ \{\langle \sigma, d'\rangle(M^\sigma_{\hat{q},d'}(s_1, \ldots, s_n)) \\ \quad | \, (\hat{q}, d') \in \mathrm{RHS}_M(q, \sigma)\} \\ \qquad (\text{それ以外}) \end{cases}$$

- $(s_1, \ldots, s_n) \in T^n_\Sigma$ および $\sigma \in \Sigma$ について，$M^\sigma_{q,d}(s_1, \ldots, s_n) = (M^\sigma_d(q(s_1)) \times \cdots \times M^\sigma_d(q(s_n)))$

ここで，$\mathrm{NS}(\sigma)$ は，σ の名前空間，$\mathrm{RHS}_M(q, \sigma) \in 2^{Q\times\Delta}$ は，R での，状態 q における σ についての遷移結果の集合である． □

次に，本稿における PDA による変換を形式的に定義する．

定義 4（PDA による変換） 定義 2 による PDA $M' = (Q', \Sigma', \Gamma', \langle q_0, \perp\rangle, R')$ による変換 $t =$

$M'([q',v](s))$ は，$s \in T_\Sigma$ から，ラベル付けされた木の集合 $t \in 2^{T_{\Sigma\times\Delta}}$ への関数として，以下のように帰納的に定義される（なお，ここで，$v \in \Gamma'^*$ は，スタックアルファベットによる文字列である）．

- $s = \sigma(s_1,\ldots,s_n)$ について，
  $M'([q',v](s)) = \{\langle\sigma,d\rangle(M'_{[\langle q,d\rangle,v']}(s_1,\ldots,s_n)) \mid (\langle q,d\rangle,v') \in \mathrm{RHS}_{M'}(q',\sigma,\mathrm{Head}(v))\}$
- $(s_1,\ldots,s_n) \in T_\Sigma^n$ について，
  $M'_{[\langle q,d\rangle,v]}(s_1,\ldots,s_n) = (M'([q'_1,v_1](s_1)) \times\cdots\times M'([q'_n,v_n](s_n)))$．ただし，$[q'_1,v_1] = [\langle q,d\rangle,v]$ であり，$[q'_{k+1},v_{k+1}] \in (\{[q'_k,v_k]\} \uparrow_{M'} s_k)$ とする．
- $s = \sigma(s_1,\ldots,s_n)$ について，
  $P \uparrow_{M'} s = P \Uparrow_{M'} \sigma \uparrow_{M'} s_1 \uparrow_{M'} \ldots \uparrow_{M'} s_n \Uparrow_{M'} \bar\sigma$ とする☆．ここで，σ は，開始タグと同一視し，$\bar\sigma$ は，σ に対応する終了タグとする．また，P は $Q' \times \Gamma'^*$ の部分集合である．
- 任意のタグ σ について $P \Uparrow_{M'} \sigma = \{[q'',v''] \mid [q',v] \in P, (q'',v') \in \mathrm{RHS}_{M'}(q',\sigma,\mathrm{Head}(v))\}$ とする．

ここで，$\mathrm{Head}(v) \in \Gamma'$ は，スタック v の先頭の文字を取り出す関数．$\mathrm{RHS}_{M'}(q,\sigma,v) \in 2^{Q\times\Delta\times\Gamma'^*}$ は，R' での，状態 q およびスタック v における σ についての遷移結果についての集合である． □

以上の定義を用いることによって，等価性は以下のように示される．

定理 1 すべての $s \in T_\Sigma$ において，$M(q_0(s)) = M'([q'_0,v_0](s))$ である．

証明（概略）：以下の木構造に関する simultanous induction によって証明できる（詳細は文献 6) 等を参照されたい）．

(i) すべての $s = \sigma(s_1,\ldots,s_n) \in T_\Sigma$ およびすべての $\sigma' \in \Sigma$ について，
$$M_d^{\sigma'}(q(s)) = M'([\langle q,d\rangle,(q,\mathrm{NS}(\sigma'))](s))$$
が成立する．

(ii) すべての $(s_1,\ldots,s_n) \in T_\Sigma^n$ および，すべての $\sigma \in \Sigma$ について，
$M^\sigma_{q,d}(s_1,\ldots,s_n) = M'_{[\langle q,d\rangle,(q,\mathrm{NS}(\sigma))]}(s_1,\ldots,s_n)$
が成立する．

(ii)⇒(i): $\mathrm{NS}(\sigma) = \mathrm{NS}(\sigma')$ のときは容易に示すことができる．$\mathrm{NS}(\sigma) \neq \mathrm{NS}(\sigma')$ のときのみ考える．

$$\begin{aligned}
M_d^{\sigma'}(q(s)) &= \{\langle\sigma,d'\rangle(M^\sigma_{q,d'}(s_1,\ldots,s_n)) \mid (\hat q,d') \in \mathrm{RHS}_M(q,\sigma)\}\\
&\quad\text{（定義 3 より）}\\
&= \{\langle\sigma,d'\rangle(M'_{[\langle\hat q,d'\rangle,(\hat q,\mathrm{NS}(\sigma))]}(s_1,\ldots,s_n)) \mid (\hat q,d') \in \mathrm{RHS}_M(q,\sigma)\}\\
&\quad\text{（(ii) より）}\\
&= \{\langle\sigma,d'\rangle(M'_{[\langle\hat q,d'\rangle,(\hat q,\mathrm{NS}(\sigma))]}(s_1,\ldots,s_n)) \mid (\langle\hat q,d'\rangle,(\hat q,\mathrm{NS}(\sigma))) \in \mathrm{RHS}_{M'}(q,\sigma,(q,\mathrm{NS}(\sigma')))\}\\
&\quad\text{（}M'\text{ の構築における 3.2.1 節の (2) より）}\\
&= M'([\langle q,d\rangle,(q,\mathrm{NS}(\sigma'))](s))\\
&\quad\text{（定義 4 より）}
\end{aligned}$$

(i)⇒(ii): まず，以下の補題を示す．

補題 1 すべての $P \subseteq Q' \times \Gamma'^*$，すべての $s \in T_\Sigma$ について，$P \uparrow_{M'} s = P$ が成立する．

補題 1 の証明：$\uparrow_{M'}$ は，union について分配則が成立するため（定義より容易に示せる），すべての $q' \in Q'$ および，すべての $v \in \Gamma'^*$ について $\{[q',v]\} \uparrow_{M'} s = \{[q',v]\}$ が成立することを示せばよい．これは，Dyck word においてタグがバランスしていることから，直観的に理解することができるが，厳密には，s について帰納法を用いて，M' の構築規則 (1)，(3) を適用することによって容易に示すことができる．

補題 1 を用いて，定理 1 の証明を続ける．

$$\begin{aligned}
&M^\sigma_{q,d}(s_1,\ldots,s_n)\\
&\quad= (M_d^\sigma(q(s_1)) \times\cdots\times M_d^\sigma(q(s_n)))\\
&\quad\quad\text{（定義 3 より）}\\
&\quad= (M'([\langle q,d\rangle,(q,\mathrm{NS}(\sigma))](s_1))\\
&\qquad\times\cdots\times M'([\langle q,d\rangle,(q,\mathrm{NS}(\sigma))](s_n)))\\
&\quad\quad\text{（(i) より）}\\
&\quad= M'_{[\langle q,d\rangle,(q,\mathrm{NS}(\sigma))]}(s_1,\ldots,s_n)\\
&\quad\quad\text{（定義 4 に補題 1 を適用し，式を比較）}
\end{aligned}$$

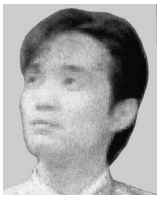
(i) を用いれば，定理 1 は以下のように示すことができる．

$$\begin{aligned}
M(q_0(s)) &= M^\omega_\perp(q(s))\\
&\quad\text{（定義 3 より）}\\
&= M'([\langle q_0,\perp\rangle,(q'_0,\mathrm{NS}(\omega))](s))\\
&\quad\text{（(i) より）}\\
&= M'([q'_0,v_0](s))\\
&\quad\text{（定義 2 より）}
\end{aligned}$$ □

(平成 18 年 9 月 15 日受付)
(平成 19 年 2 月 27 日採録)

(担当編集委員 石川 博，有次 正義，片山 薫，木俵 豊，中島 伸介)

☆ s は木構造であるので，この帰納的定義は well-defined である．



宮下 尚（正会員）

2001 年 Free Standards Group, OpenI18N WG で Input Method Subgroup Leader として入力メソッドの開発および標準化に従事．2003 年日本 IBM 東京基礎研究所に入社．以来，XML，高信頼性ミドルウェア，アクセシビリティの研究開発に従事．ACM 会員．

村田 真（正会員）

日本 IBM（株）東京基礎研究所特別研究員・国際大学研究所特任研究員．1982 年京都大学理学部卒業．W3C，IETF，OASIS，ISO/IEC，国内委員会において XML の仕様制定・研究に従事．インターネットコンファレンス’98 論文賞．情報処理学会業績賞受賞．