UMS 仕様書

Version 0.4 Draft 27 Jul 2006

このバージョン:

Version 0.4 Draft 27 Jul 2006

前のバージョン:

Version 0.4 Draft 14 Jul 2006

編者:

松崎 恵一 <matuzaki@plain.isas.jaxa.jp>

訳者:

松崎恵一, 宇宙航空研究開発機構

RELAX NG 仕様書

Committee Specification 3 December 2001

このバージョン:

Committee Specification: 3 December 2001

前のバージョン:

Committee Specification: 11 August 2001

編者:

James Clark <jjc@jclark.com>, 村田 真 <EB2M-MRT@asahi-net.or.jp>

訳者:

小町祐史, 松下電送(株)
川俣晶, (株)ピーデー
山本陽平, リコー(株)
村田真, 国際大学

本文書は RELAX NG の仕様書と UMS の仕様書を併記した文書である。

緑色文字の部分と**黒色文字**の部分をあわせたのが OASIS による RELAX NG の仕様書である。

青色文字の部分と**黒色文字**の部分をあわせたのが UMS Group による UMS の仕様書である。

仕様には含まれない読者のための注釈を**橙色文字**で示す。

注意

この翻訳は，$Date: 2003/07/26 02:00:04 $に作成した。

改定履歴

20050919 line → lines 機能変更

20060122 List 要素の delimiter および separator 属性には C および JAVA の文字リテラルの形式で区切り文字を指定することとした、仕様には未記載。従来の lines は list delimiter="¥n" で置き換え可能なため削除した。

20060328 encode 及び文字列の方針を決定し記述に取り込んだ。

20060330 length, separator, delimiter の型をそれぞれ int, string, string に訂正。

20060408 separate, delimit の意味を明確化。

20060419 "撤回されることがない" 条項をとりあえず削除。

20060714 7.2 節 – list を simple() から complex() に変更。その他 7.2 節の誤記訂正、付加説明追加。

20060714 3 章 – encode 属性の値を規定。

20060727 3 章 – 誤記訂正（ns 属性に関する事項の削除もれ）

20060727 3 章 EscapeSequence – 文字コードの記法を仕様書中の他の箇所で使われている #xNN に変更

20060727 6.2.4 節 – UMS における text の意味を（列ではない）文字列に限定

20060727 7.1.5 節 – start 下の list を禁止（厳密に言うと 6.2.7 と冗長）

20060727 7.2 節 – 誤記訂正（要素→bit, byte）

20060727 7.2 節（text）– UMS における <text /> の内容種別を simple に変更

概要

この規格は，XML のための簡潔なスキーマ言語 RELAX NG UMS を規定する。RELAX NG は、[RELAX]及び[TREX]に基づく。UMS は、RELAX NG に基づく。RELAX NG UMS スキーマは，XML 文書の構造及び内容に関するパターンを規定する。 RELAX NG UMS スキーマは，それ自体が XML 文書になる。

この文書の状態

この委員会規定は，OASIS RELAX NG 技術委員会によってその公表が承認された。この規定は安定した文書であって，委員会の合意に基づく。この規定に関するコメントは，relax-ng-comment@lists.oasis-open.org へ送られたい。

この規定についての既知の誤りは，
http://www.oasis-open.org/committees/relax-ng/spec-20011203-errata.html に掲載してある。

この規定は UMS Version 0.4 の仕様策定に向けた草稿である。

注意

2003 年 7 月 19 日時点での正誤表は、この日本語訳にはすべて反映されている。

目次

1. はじめに

この規格は，次を規定する。

- XML 文書が正しい RELAX NG UMS スキーマになるのはどんな場合か
- 正しい RELAX NG UMS スキーマに照らして、XML 文書が妥当になるのはどんな場合か

正しい RELAX NG UMS スキーマに照らして妥当性検証される XML 文書を，インスタンスと呼ぶ。

この規格の構成を次に示す。**項 2** は，データモデルを示し，この規格の以降の部分で用いる XML 文書の抽象化を示す。**項 3** は，RELAX NG UMS スキーマの構文を示す。どんな正しい RELAX NG UMS スキーマも，この構文に適合しなければならない。**項 4** は，RELAX NG UMS スキーマを単純化するために適用される変換の並びを示す。これらの変換には，正しい RELAX NG UMS スキーマが満たさなければならないある制限をチェックすることも含む。**項 5** は，変換を適用した結果の構文を示す。この単純な構文は，完全な構文の部分集合である。**項 6** は，単純な構文を用いた正しい RELAX NG UMS スキーマの意味を示す。ここで意味とは，要素が RELAX NG UMS スキーマに照らして妥当になるのはどんなときかの規定である。**項 7** は，単純な構文を用いて制限事項を示す。単純な形式への変換の後に、正しい RELAX NG UMS スキーマはこれらの制限事項を満たさなければならない。最後に**項 8** は，RELAX NG UMS の妥当性検証器に関する適合性要件を示す。

この規格とは別の入門書が用意されている。([Tutorial]を参照されたい。)

UMS のスキーマ（mapping definition ではなく）の入門書が必要。

2. データモデル

UMS では，インスタンスを表現する(単なる)文書は，ビット列から構成される。

RELAX NG UMS では，スキーマとインスタンスのどちらを表現する XML 文書もを抽象データモデルによって扱う。スキーマやインスタンスを表現する XML 文書は，[XML 1.0]に適合する整形式でなければならず，[XML Namespaces]の制約に適合しなければならない。

XML 文書は，要素によって表現される。要素は，次のものから成る。

- 名前
- 文脈
- 属性の集合
- 0 個以上の子の順序付き並び（どの子も，要素又は空でない文字列のどちらかとし，並びの中に二つの文字列が続いて現れてはならない。）

名前は，次のものから成る。

- 名前空間 URI を表す文字列（空文字列は特別な意味をもち，名前空間がないことを表す。）
- 局所名を表す文字列（この文字列は，[XML Namespaces]の NCName 生成規則とマッチする。）

文脈は，次のものから成る。

- 基底 URI
- 名前空間マップ。これは，接頭辞を名前空間 URI に対応付ける。また，デフォルト名前空間 URI（xmlns 属性によって宣言されるもの。）を指定してもよい。

属性は，次のものから成る。

- 名前
- 値を表す文字列

文字列は，0 個以上の文字の並びから成り，文字は[XML 1.0]において定義される。

XML 文書のための要素は，次に示すとおりに[XML Infoset]のインスタンスから構成される。情報項目の *x* 特性の値を参照するために，記法 [*x*]を用いる。要素を文書情報項目から構成するには，[document element]から要素を構成する。要素を要素情報項目から構成するには，名前を[namespace name]及び[local name]から構成し，文脈を[base URI]及び [in-scope namespaces]から構成し，属性（複数）を[attributes]から構成し，子（複数）を[children]から構成する。要素の属性（複数）を属性情報項目の順序なし集合から構成するには，各属性情報項目から属性を構成することを繰り返す。要素の子（複数）を子情報項目のリストから構成するには，まず要素情報項目及び文字情報項目以外の情報項目を削除し，次にリスト中の各要素情報項目から要素を構築し，連続する文字情報項目をできるだけ繋げることによって文字列を構成する。属性を属性情報項目から構成するには，名前を[namespace name]及び [local name]から構成し，値を[normalized value]から構成する。要素及び属性の名前を[namespace name]及び[local name]から構成するとき，[namespace name]特性が存在しなければ，空の文字列及び[local name]から構成する。文字列を文字情報項目の並びから構成するには，各文字情報項目の[character code]から文字を構成することを繰り返す。

単一の XML 文書について複数の異なる情報集合があり得る。これは，すべての DTD 宣言を処理し，すべての外部解析対象実体を展開することが XML パーサに義務付けられていないことによる。これらの複数の情報集合の中には，[all declarations processed]が真であり，未拡張実体参照情報項目を全く含まない情報集合が正確に一つだけ存在する。この情報集合を，RELAX NG UMS データモデルを定義するための基本とする。

2.1. 例

文書を表すビット列が，次のユニコード文字(UTF8)の並びからなる。

- ‘A’
- ‘B’
- ‘C’

このビット列のバイト長は3である。

XML 文書 http://www.example.com/doc.xml が，次に示すものとする。

```
<?xml version="1.0"?>
<foo><pre1:bar1 xmlns:pre1="http://www.example.com/n1"/><pre2:bar2
  xmlns:pre2="http://www.example.com/n2"/></foo>
```

この文書を表す要素は，次のものをもつ。

- 名前（これは，次のものをもつ。）
  - 名前空間 URI としての空文字列（名前空間がないことを表す。）
  - 局所名としての foo
- 文脈（これは，次のものをもつ。）
  - 基底 URI として http://www.example.com/doc.xml
  - 名前空間マップ（これは，次のとおりとする。）
    - 接頭辞 xml を名前空間 URI http://www.w3.org/XML/1998/namespace にマップする。(xml 接頭辞は，すべての XML 文書において暗黙的に宣言される。）
    - デフォルトの名前空間 URI として空文字列を指定する。
- 属性の空集合
- 次のものをもつ要素から成る子どもの並び
  - 名前（これは，次のものをもつ。）

        - 名前空間 URI としての http://www.example.com/n1
        - 局所名として bar1
    - 文脈（これは，次のものをもつ。）
        - 基底 URI としての http://www.example.com/doc.xml
        - 名前空間マップ（これは，次のとおりとする。）
            - 接頭辞 pre1 を名前空間 URI http://www.example.com/n1 にマップする
            - 接頭辞 xml を名前空間 URI http://www.w3.org/XML/1998/namespace にマップする。
            - デフォルトの名前空間 URI として空文字列を指定する。
    - 属性の空集合
    - 子どもの空の並び

これらには，次のものをもつ要素が続く。

- 名前（これは，次のものをもつ。）
    - 名前空間 URI としての http://www.example.com/n2
    - 局所名としての bar2
- 文脈（これは，次のものをもつ。）
    - 基底 URI としての http://www.example.com/doc.xml
    - 名前空間マップ（これは，次のとおりとする。）
        - 接頭辞 pre2 を名前空間 URI http://www.example.com/n2 にマップする。
        - 接頭辞 xml を名前空間 URI http://www.w3.org/XML/1998/namespace にマップする。
        - デフォルトの名前空間 URI として空文字列を指定する。
- 属性の空集合
- 子(複数)の空の列

注意

UMS では、XML 文書をスキーマのみに用いる。この例は一般的な XML 文書であり、UMS のスキーマではない。

3. 完全な構文

次の文法は，RELAX NG UMS の構文を要約する。ここでの記法は， RELAX NG UMS スキーマを XML で表現したものを，文字の並びとして扱う。しかし、文法はデータモデルレベルのものであることを理解しなければならない。例えば，ここでの構文は<text/>を使用しているが，インスタンス又はスキーマでは <text></text>を代わりに使用できる。これは，データモデルレベルでは，どちらも同じ要素を表すことによる。この文法に示されるすべての要素は，次に示す名前空間 URI で修飾される。

http://relaxng.org/ns/structure/1.0
http://ums.isas.jaxa.jp/0.4

記号 QName 及び NCName は，[XML Namespaces]において定義される。anyURI 記号は，[W3C XML Schema Datatypes]の anyURI データ型と同じ意味をもつ。すなわち，文字列であって，許可されない値を [XLink]の 5.4 に示されるとおりに別扱いすると，[RFC 2396]（これは[RFC 2732]によって修正されている。）に定義される URI 参照になるものを示す。記号 string は，どんな文字列ともマッチする。

明示的に示される属性に加えて，どの要素も ns 属性をもつことができ，どの要素も datatypeLibrary 属性をもつことができる。 ns 属性は，どんな値をももつことができる。 datatypeLibrary 属性の値は，前の段落に示される anyURI 記号とマッチしなければならない。さらに，URI 参照の相対形式を用いてはならず，素片識別子をもってはならない。例外として，この属性の値は空文字列であってもよい。

どの要素も，文法に示される属性に加えて，外来の属性をもつこともできる。外来の属性とは，空文字列でも RELAX NG UMS 名前空間 URI でもない名前空間 URI の名前をもつ属性とする。文字列である子をもつことができないどの要素も(つまり，value， param 及び name 以外のどの要素も)，文法に示される子要素に加えて，外来の子要素をもってもよい。外来の要素は， RELAX NG UMS 名前空間 URI でない名前空間 URI の名前をもつ要素とする。他の子要素と外来の子要素の間の相対的な位置についての制約はない。

どの要素も，子として，完全に空白文字のみから成る文字列をもつことができる。ここで空白文字は，#x20，#x9，#xD 又は#xA の一つとする。空白文字列である子と子要素の間の相対的な位置についての制約はない。

name，type 及び combine の各属性の値，並びに各 name 要素の内容は、先頭及び末尾に空白をもってもよい。

pattern ::= <element name="*QName*"> *pattern*+ </element>
| <element> *nameClass pattern*+ </element>
| <attribute name="*QName*"> [*pattern*] </attribute>
| <attribute> *nameClass* [*pattern*] </attribute>
<byte [encode="*Encode*"] [length="*Length*"]> *pattern** </byte>
| <byte [encode="*Encode*"]> *lengthClass pattern** </byte>
| <bit [encode="*Encode*"] [length="*Length*"]> *pattern** </bit>
| <bit [encode="*Encode*"]> *lengthClass pattern** </bit>
| <group> *pattern*+ </group>
| <interleave> *pattern*+ </interleave>
| <choice> *pattern*+ </choice>
| <optional> *pattern*+ </optional>
| <zeroOrMore> *pattern*+ </zeroOrMore>
| <oneOrMore> *pattern*+ </oneOrMore>
| <list> *pattern*+ </list>
| <list separator="*Char*"> *pattern*+ </list>
| <list delimiter="*Char*"> *pattern*+ </list>
| <mixed> *pattern*+ </mixed>
| <ref name="*NCName*"/>
| <parentRef name="*NCName*"/>
| <empty/>
| <text/>
| <value [type="*NCName*"]> *string* </value>
| <data type="*NCName*"> *param** [*exceptPattern*] </data>
| <notAllowed/>
| <externalRef href="*anyURI*"/>
| <grammar> *grammarContent** </grammar>

Param ::= <param name="*NCName*"> *string* </param>

exceptPattern ::= <except> *pattern*+ </except>

grammarContent ::= *start*
| *define*
| *grammarContent**
| <include href="*anyURI*"> *includeContent** </include>

includeContent ::= *start*
| *define*
| *includeContent**

start ::= <start [combine="*method*"]> *pattern* </start>

define ::= <define name="*NCName*" [combine="*method*"]> *pattern*+ </define>

method ::= choice
| interleave

```
nameClass          ::= <name> QName </name>
                     | <anyName> [exceptNameClass] </anyName>
                     | <nsName> [exceptNameClass] </nsName>
                     | <choice> nameClass+ </choice>
exceptNameClass    ::= <except> nameClass+ </except>
lengthClass        ::= <length> QName </length>
                     | <anyLength> [exceptLengthClass] </anyLength>
                     | <choice> lengthClass+ </choice>
exceptLengthClass  ::= <except> lengthClass+ </except>
```

separator, delimiter 属性の値は以下に定義される。

```
Char             ::= anyCharacter
                   | EscapeSequence
EscapeSequence   ::=  ¥ b          /* #x8: 後退 BS */
                   | ¥ t          /* #x9: 文字タブ HT */
                   | ¥ n          /* #xA: 改行 LF */
                   | ¥ f          /* #xC: 書式送り FF */
                   | ¥ r          /* #xD: 復帰 CR */|
                   | ¥ ″          /* #x22: 引用符″ */
                   |¥ ’           /* #x27: アポストロフィー’*/
                   |¥ ¥           /* #x5c: 逆斜線¥ */
                   | OctalEscape  /* #x0 から#xFF まで。*/
OctalEscape      ::=  ¥ OctalDigit
                   | ¥ OctalDigit OctalDigit
                   | ¥ ZeroToThree OctalDigit OctalDigit
OctalDigit       ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
ZeroToThree      ::=  0 | 1 | 2 | 3
```

encode 属性の値は以下に定義される。

```
Encode ::=  txt | signed | unsigned | ieee754
```

将来的には encode library が必要となるだろう。上記は組み込みの encode library にあたるもの。

3.1. 例

項 2.1 の文書のためのスキーマ例を完全な構文で示す。

```
<?xml version="1.0"?>
<element name="foo"
         xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:a="http://relaxng.org/ns/annotation/1.0"
         xmlns:ex1="http://www.example.com/n1"
         xmlns:ex2="http://www.example.com/n2">
  <a:documentation>A foo element.</a:documentation>
  <element name="ex1:bar1">
    <empty/>
  </element>
```

```
  <element name="ex2:bar2">
    <empty/>
  </element>
</element>

<?xml version="1.0"?>
<byte encode="txt"
      xmlns="http://ums.isas.jaxa.jp/0.4"
      xmlns:a="http://ums.isas.jaxa.jp/0.4/annotation">
  <a:documentation>A foo element.</a:documentation>
  <byte length="1"/>
  <byte length="2"/>
</byte>
```

4. 単純化

前の節で与えられた完全な構文は、以下の変換規則を順番とおりに適用することで、より単純な構文に変換する。それぞれの規則は、次の規則が適用される前に、スキーマ内の全ての要素に適用されたかのような効果をもたらさねばならない。個々の変換規則は、正しいスキーマが満たすべき制約を指定している場合がある。変換規則はデータモデルのレベルで適用する。スキーマを解析し、データモデルのインスタンスを構築した後に変換を適用する。

4.1. 注釈

外来の属性および要素は取り除かれる。

注意

この段階で xml:base を取り除くことは安全である。なぜなら, xml:base 属性は要素情報項目の[base URI]を確定するために用いられ、さらに[base URI]から要素の文脈の基底 URI が作られている。したがって、文書がデータモデルのインスタンスに解析後に xml:base 属性を処分することができる。

4.2. 空白

value 及び param を除くどの要素についても、空白文字のみからなる文字列である子を取り除く。

name length, type および combine 属性の値と、name length 要素の内容において、先頭および末尾の空白文字を取り除く。

4.3. datatypeLibrary 属性

datatypeLibary 属性の値は、[XLink]の 5.4 節で指定されるとおりに、許容されない文字を別扱いすることによって変換する。

datatypeLibrary 属性をもたないどの data 又は value 要素にも、datatypeLibrary 属性を追加する。追加される datatypeLibrary 属性の値は、datatypeLibrary 属性をもつもっとも近い祖先要素の datatypeLibrary 属性の値とする。そのような祖先が存在しない場合は空文字列とする。つぎに、datatypeLibrary 属性であって、data または value 以外の要素に付属するものを取り除く。

4.4. value 要素の type 属性

type 属性を持たない value 要素には，値が token である type 属性を付け加え，datatypeLibrary 属性の値を空文字列に置き換える。

4.5. href 属性

externalRef または include 要素の href 属性の値は、まず[XLink]の 5.4 節で指定されるとおりに、許されない文字を別扱いすることによって変換する。次に，URI 参照を[RFC 2396]の 5.2 に記述された絶対形式に帰着させる。これは、href 属性を持つ要素の文脈から得られる基底 URI を用いて行う。

href 属性の値は（**項 2** で示されたとおりに）要素を構築するために使用する。これは、以下のように行われねばならない。URI 参照は、URI そのものと素片識別子からなる（素片識別子は省いてもよい）。まず URI によって特定される資源を取り出す。その結果は MIME エンティティであり、MIME メディア型がラベルとして付されたバイト列になる。MIME エンティティと素片識別子（なくてもよい）から要素がどのように構築されるかは、メディア型によって定まる。メディア型が、application/xml または text/xml であるとき、関連する RFC（執筆時点では[RFC 3023]）に従って、MIME エンティティは XML 文書として解析されねばならない。そして、解析結果から，**項 2** で指定されたように、要素を構築しなければならない。特に、 charset パラメタはこの RFC で指定されたとおりに扱われねばならない。この仕様書は、application/xml と text/xml を除き、メディア型の取り扱いを定義しない。 href 属性は、素片識別子を含んではならない。ただし、その属性によって特定される資源のもつメディア型の登録が、そのメディア型のための素片識別子の解釈のしかたを定義している場合は、含んでもよい。

注意

[RFC 3023] は application/xml または text/xml の素片識別子の解釈を定義しない。

4.6. externalRef 要素

externalRef 要素は、以下のとおりに変換する。 **項 4.5** で指定されたとおりに，href 属性の値である URI 参照を用いて要素を構築する。この要素は、パターンの構文にマッチしなければならない。この要素を、この節で与えた規則及び本章の以前の節で与えた規則を適用して再帰的に変換する。この変換は、繰り返しに陥ってはならない。言い換えれば、参照された要素を変換するとき、同じ値の href 属性をもつ externalRef 要素の参照が起こってはならない。

参照された要素が ns 属性を持たないなら、 externalRef 要素の ns 属性を、参照された要素に転記する。そして、externalRef 要素を、参照された要素と置き換える。

4.7. include 要素

include は以下のとおりに変換する。**項 4.5** で指定されたとおりに、href 属性の値を URI 参照として用いて要素を構築する。この要素は、文法の構文にマッチする grammar 要素でなければならない。

この grammar 要素は、本節で与えた規則及び本章の以前の節で与えた規則を適用して再帰的に変換する。この変換は、繰り返しに陥ってはならない。言い換えれば、grammar 要素を変換するとき、同じ値の href 属性をもつ include 要素の参照が起こってはならない。

ある要素の*構成部品*とは、その要素のすべての子（複数）及びすべての div 子要素の構成部品であると定義する。 include 要素が start 構成部品をもつなら、grammar 要素は start 構成部品を持たねばならない。include 要素が start 構成部品をもつなら、すべての start を grammar 要素から取り除く。 include 要素が

define 構成部品をもつなら、grammar 要素は同じ名前の define 構成部品を持たねばならない。 include 要素のどの define 構成部品についても、同じ名前のすべての define 構成部品を grammar 要素から取り除く。

include 要素を div 要素に変換する。まず、 div の属性は、href 属性を除き、include の属性とする。 div 要素の子(複数)は、(直前の段落で述べたとおりに start と define 構成部品を取り除いた後の)grammar 要素であり、その後に include 要素の子(複数)が続く。次に, grammar 要素を div 要素に変換する。

4.8. element と attribute 要素の name 属性

element または attribute 要素の name 属性は name 子要素に変換する。

もし attribute 要素が name 属性を持ち ns 属性を持たないなら、ns=""属性を name 子要素に追加する。

4.8. byte と bit 要素の length 属性

byte または bit 要素の length 属性は length 子要素に変換する。

4.9. ns 属性

ns 属性を持たない name, nsName または value 要素には、ns 要素を追加する。追加される ns 属性の値は、ns 属性をもつもっとも近い祖先要素の ns 属性の値とする。そのような祖先要素が存在しない場合は空文字列とする。次に、ns 属性であって、name, nsName または value 以外の要素にあるものを取り除く。

注意

ns 属性の値には、許されない文字を別扱いする変換又は他のいかなる変換も*適用しない*。なぜなら、インスタンス中の名前空間 URI にはどのような変換も適用されておらず、ns 属性の値はこの名前空間 URI と比較されるからである。

注意

include と externalRef 要素が解決されるのは、datatypeLibrary 属性が追加された後で、ns が追加される前なので、ns 属性は外部スキーマにも継承されるが、datatypeLibrary 属性はそうではない。

4.10. QName

接頭辞をもつどの name 要素についても、接頭辞を取り除き、ns 属性を追加する。既存の ns 属性があっても置き換える。追加する ns 属性の値は、name 要素の文脈での名前空間マップによってこの接頭辞がマップされるものとする。文脈は、この接頭辞についてのマップを持たなければならない。

4.11. div 要素

div 要素は、その子(複数)によって置き換える。

4.12. 子要素の数

define, oneOrMore, zeroOrMore, optional, list または mixed 要素は、正確に 1 つの子要素をもつように変換する。それが 1 つより多くの子要素をもつなら、それらを group 要素によってまとめる。同様に、element bit または byte 要素は正確に 2 つの子要素をもつように変換する。最初のものは名前長さクラスで、2 番目のものはパ

ターンである。 bit または byte 要素が長さクラスを子要素に持たない場合 anyLength 要素を1番目の子要素に追加する。2 つより多くの子要素をもつなら、最初の要素以外の子要素を group 要素でまとめる。

except 要素は正確に 1 つの子要素をもつように変換する。 1 つより多くの子要素をもつなら、子要素を choice 要素でまとめる。

attribute bit または byte 要素がたった 1 つの子要素(名前長さクラス)をもつなら、text 要素を追加する。

choice, group または interleave 要素は正確に 2 つの子要素をもつように変換する。 1 つの子要素をもつなら、この子要素によって置き換える。2 つより多くの子要素をもつなら、親要素と同じ名前の新しい要素によって最初の 2 つの子要素をまとめる。たとえば、

<choice> *p1 p2 p3* </choice>

は以下のように変換する。

<choice> <choice> *p1 p2* </choice> *p3* </choice>

この操作は子要素の数を 1 つ減らす。子要素の数が正確に 2 つになるまで、この変換を繰り返し適用する。

4.13. mixed 要素

mixed 要素は text 要素を伴うインタリーブに変換する。

<mixed> *p* </mixed>

は以下のように変換する。

<interleave> *p* <text/> </interleave>

4.14. optional 要素

optional 要素は、choice 要素に変換する。この choice 要素の片方の子は, optional 要素の子とし, もう一方の子は empty とする。

<optional> *p* </optional>

は以下のとおりに変換する。

<choice> *p* <empty/> </choice>

4.15. zeroOrMore 要素

zeroOrMore 要素は choice 要素に変換する。この choice 要素の片方の子は, <oneOrMore> *p* </oneOrMore> 要素であり, もう一方の子は empty 要素とする。ここで *p* は zeroOrMore 要素の子とする。

<zeroOrMore> *p* </zeroOrMore>

は以下のとおりに変換する。

<choice> <oneOrMore> *p* </oneOrMore> <empty/> </choice>

4.16. 制約

この規則は、変換は行わないが、様々な制約をチェックする。

注意

この節の制約は、項 7 で規定される制約と異なり、いかなる ref 要素も解決することなしにチェックすることができる。そのため、これらの制約は，到達可能ではない（項 4.19 参照）という理由または notAllowed(項 4.20 参照)が原因でこのあとの段階の単純化の間には現れないパターンにも適用される。

anyName 要素の子要素である except 要素は、 anyName 子孫要素をもってはならない。 nsName 要素の子要素である except 要素は、 nsName または anyName 子孫要素をもってはならない。

anyLength 要素の子要素である except 要素は、 anyLength 子孫要素をもってはならない。

attribute 要素の最初の子または attribute の最初の子の子孫として出現し、ns 要素が空文字列と等しい name 要素は、xmlns と等しい内容を持ってはならない。

attribute 要素の最初の子または attribute の最初の子の子孫として出現する name または nsName 要素は、http://www.w3.org/2000/xmlns という値の ns 属性を持ってはならない。

注意

[XML Infoset]は名前空間宣言属性の名前空間 URI が http://www.w3.org/2000/xmlns であるべきことを定義している。

data または value 要素は、エンコード型及びデータ型を正しく使用しなければならない。特に、type 属性は、datatypeLibrary 属性の値で識別されるデータ型ライブラリに含まれるエンコード型及びデータ型を特定しなければならない。 data 要素において、パラメタリストはデータ型（項 6.2.8 参照）で許されるものの 1 つでなければならない。

**** エンコード型をどう参照するのか記述が必要、そもそも RELAX NG においてこのチェックは制約として必要なのか？意味と冗長に思える。 ****

4.17. combine 属性

個々の grammar 要素において、同じ名前をもつすべての define 要素はそれぞれ結合される。 どの名前についても、その名前をもつ define 要素であって combine 属性を持たないものが 1 つより多く存在してはならない。 どの名前に対しても、その名前をもつ define 要素であって combine 属性の値が choice なものがあれば、その名前をもつ define 要素であって combine 属性の値が interleave なものがあってはならなない。 したがって、どの名前についても、その名前をもつ define 要素が 1 つより多く存在するなら、その名前に対する combine 属性の値は一意に定まる。この一意な値を確認した後で、combine 属性は取り除かれる。以下の二つの定義

```
<define name="n">
  p1
</define>
<define name="n">
  p2
</define>
```

は以下のとおりに結合される。

```
<define name="n">
  <c>
    p1
    p2
  </c>
</define>
```

ここで *c* は combine 属性の値である。個々の名前に対して正確に 1 つの define 要素になるまで、define 要素の対を結合することを繰り返す。

同様に、個々の grammar 要素は、すべての start 要素を互いに結合する。 combine 属性を持たない start 要素が 1 つより多く存在してはならない。もし、choice という値の combine 属性をもつ start 要素が存在するなら、interleave という値の combine 属性をもつ start 要素が存在してはならない。

4.18. grammar 要素

この規則によって、スキーマはそのトップレベル要素が grammar であり、それ以外の grammar 要素は存在しないように変換される。

ある要素の*直上の文法*とは、最も近い祖先の grammar 要素であると定義する。 ref 要素が define 要素を *参照する*のは、それらの name 属性の値が同じで、それらの直上の文法が同じときである。 parentRef 要素が define 要素を *参照する*のは、それらの name 属性の値が同じであり、 parentRef 要素の直上の文法の直上の文法が、 define 要素の直上の文法と同じときである。 どの ref または parentRef 要素も、 define 要素を参照しなければならない。 grammar 要素は一つの start 子要素をもたなければならない。

最初に、トップレベルのパターン *p* を、<grammar><start>*p*</start></grammar>に置き換える。次に、define 要素の名前を、スキーマ内のいずれの 2 つの define も同じ名前を持たないように変える。define 要素の名前を変更するには、name 属性の値を変更し、その define 要素を参照するすべての ref 及び parentRef 要素の name 属性の値を変更する。次に、すべての define 要素をトップレベルの grammar 要素の子になるように移動させる。ネストした個々の grammar 要素をその start 要素の子で置き換え、個々の parentRef 要素を ref 要素に名前を変更する。

4.19. define 及び ref 要素

この規則では、文法は element bit 及び byte 要素はいずれも define 要素の子となり、 define 要素の子はいずれも element bit もしくは byte 要素となるように変換される。

最初に、*到達可能*ではない define 要素を取り除く。 define 要素は、到達可能な ref 要素に参照されているときに到達可能である。 ref 要素は、start 要素または到達可能な define 要素の子孫であるとき、到達可能である。いま、define 要素の子ではない個々の element bit 及び byte 要素について、grammar element に define 要素を追加する。そして、element 要素を ref 要素で置き換え、追加した define を参照する。 追加される define 要素の name 属性の値は、他のいかなる define 要素のもつ name 属性の値とも異なっていなければならない。 追加される define 要素の子要素は、element bit もしくは byte 要素である。

ref 要素が*展開可能*なのは、その参照する define 要素が、element bit もしくは byte 要素を子としてもたないときと定義する。 展開可能な ref 要素であって、 start 要素又は element 要素の子孫であるものは展開する。展開をするには、参照されている define 要素の子要素で ref 要素を置き換え、この置き換え後に含まれる展開可能な ref 要素を再帰的に展開する。この結果ループが発生してはならない。言い換えれば、 *n* という値の name をもつ ref 要素の置き換えを展開するときに、再び *n* という値の name をもつ ref 要素の展開を必要としてはならない。最後に、子が element bit もしくは byte 要素ではないすべての define 要素を取り除く。

4.20. notAllowed 要素

この規則では、文法は notAllowed 要素が start 又は element bit もしくは byte 要素の子要素としてしか出現しないように変換する。 notAllowed 子要素をもつ attribute, list, group, interleave, または oneOrMore 要素は、notAllowed に変換する。 2 つの notAllowed 子要素をもつ choice 要素は、 1 つの notAllowed 要素に変換する。 1 つの notAllowed 子要素をもつ choice 要素の子要素は、もう一方の子要素に変換する。 notAllowed 子要素をもつ except 要素は、取り除く。 これらの変換は、適用できなくなるまで繰り返し適用する。 到達可能ではない define 要素は取り除く。

4.21. empty 要素

この規則では、文法は empty 要素が group, interleave, または oneOrMore 要素の子要素または choice 要素の 2 番目の子要素に出現しないように変換する。 2 つの empty 子要素を group, interleave 又は choice 要素は 1 つの empty 要素に変換する。 1 つの empty 子要素をもつ group 又は interleave 要素は、もう一方の子要素に変換する。 2 番目の子要素が empty 要素である choice 要素は 2 つの子要素を交換することによって変換する。 empty 子要素をもつ oneOrMore 要素は empty 要素に変換される。 これらの変換は、適用できなくなるまで繰り返し適用する。

5. 単純な構文

項 4 にあるすべての規則を適用すれば、スキーマは次の文法にマッチすることになる。

```
grammar          ::= <grammar> <start> top </start> define* </grammar>
define           ::= <define name="NCName"> <element> nameClass top </element> </define>
                     <define name="NCName"> <byte [encode="Encode"]> lengthClass top </byte>
                     </define>
                     | <define name="NCName"> <bit [encode="Encode"]> lengthClass top </bit>
                     </define>
top              ::= <notAllowed/>
                     | pattern
pattern          ::= <empty/>
                     | nonEmptyPattern
nonEmptyPattern  ::= <text/>
                     | <data datatypeLibrary="anyURI" type="NCName"> param* [exceptPattern]
                     </data>
                     | <value datatypeLibrary="anyURI" type="NCName" ns="string"> string </value>
                     | <list> pattern </list>
                     | <list separator="Char"> pattern </list>
                     | <list delimiter="Char"> pattern </list>
                     | <attribute> nameClass pattern </attribute>
                     | <ref name="NCName"/>
                     | <oneOrMore> nonEmptyPattern </oneOrMore>
                     | <choice> pattern nonEmptyPattern </choice>
                     | <group> nonEmptyPattern nonEmptyPattern </group>
                     | <interleave> nonEmptyPattern nonEmptyPattern </interleave>
param            ::= <param name="NCName"> string </param>
exceptPattern    ::= <except> pattern </except>
nameClass        ::= <anyName> [exceptNameClass] </anyName>
                     | <nsName ns="string"> [exceptNameClass] </nsName>
```

```
                    | <name ns="string"> NCName </name>
                    | <choice> nameClass nameClass </choice>
exceptNameClass  ::= <except> nameClass </except>
lengthClass      ::= <anyLength> [exceptLengthClass] </anyLength>
                    | <length> QName </length>
                    | <choice> lengthClass+ </choice>
exceptLengthClass ::= <except> lengthClass </except>
```

この文法では、明示的に許容されていない要素や属性は許容されない。

5.1. 例

次の例は、 項 3.1 にあるスキーマが単純な構文にどう変換されるかを示す。

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <ref name="foo.element"/>
  </start>

  <define name="foo.element">
    <element>
      <name ns="">foo</name>
      <group>
        <ref name="bar1.element"/>
        <ref name="bar2.element"/>
      </group>
    </element>
  </define>

  <define name="bar1.element">
    <element>
      <name ns="http://www.example.com/n1">bar1</name>
      <empty/>
    </element>
  </define>

  <define name="bar2.element">
    <element>
      <name ns="http://www.example.com/n2">bar2</name>
      <empty/>
    </element>
  </define>
</grammar>

<?xml version="1.0"?>
<grammar xmlns="http://ums.isas.jaxa.jp/0.4">
  <start>
    <ref name="byte1"/>
  </start>
```

```
  <define name="byte1">
    <byte encode="txt">
      <anyLength/>
      <group>
        <ref name="byte2"/>
        <ref name="byte3"/>
      </group>
    </byte>
  </define>

  <define name="byte2">
    <byte>
      <length>1</length>
      <text/>
    </byte>
  </define>

  <define name="byte3">
    <byte>
      <length>2</length>
      <text/>
    </element>
  </byte>
</grammar>
```

注意

厳密に言えば，単純化の結果はデータモデルに従うインスタンスであって XML 文書ではない。簡便さのため、データモデルに従うインスタンスを XML 文書によって表現する。

6. 意味

この章では、単純な構文に変換された正しい RELAX NG UMS スキーマの意味を定義する。RELAX NG UMS スキーマの意味は，そのスキーマに照らして妥当な XML 文書がなんであるかという指定である。意味は形式的に記述する。形式化には，公理と推論規則を用いる。公理は，無条件に証明可能な命題である。推論規則は，ひとつ以上の前提条件をもち、正確に一つの結論をもつ。前提条件は，肯定的又は否定的である。推論規則の肯定的な前提条件のすべてが証明可能で、否定的な推論規則のいずれも証明可能ではないとき、推論の結論は証明可能である。XML 文書が RELAX NG UMS スキーマに照らして妥当であるのは、それが妥当であるという命題が本章で示す形式化において証明可能なときである。

注意

ここで用いた形式化は証明システムに類似する。しかし、伝統的な証明システムは肯定的な前提条件しか持たない。

推論規則のための記法は、前提条件と結論を水平線で分ける。前提条件は線の上に、結論は線の下に示す。前提条件が $\mathrm{not}(p)$ の形をしている場合は，否定的な前提条件である。そうでなければ、肯定的な前提条件である。公理も推論規則も変数を用いることができる。変数は名前をもち、さらに添え字をもつこともある。変数の名前はイタリックで示す。変数の名前から、その変数がとりうる値が決まる。公理と推論規則には，それが含む変数についての全称記号が暗黙に適用される。これについては、次に詳しく説明する。

一つの推論規則又は公理において、ある変数が 1 回以上出現することがありうるので、変数がとりうるオブジェクトごとに同値関係を定義する必要がある。どの種類のオブジェクトについても、同値関係は値に基づいて決まる。ある種類の二つのオブジェクトが同値であるのは、それらの構成物が同値であるときである。例えば，二つの属性が同じとみなすのは、それらの名前も値も同じときである。二つの文字が同値であるのは、その Unicode 文字コードが同じときである。

6.1. 名前クラス

名前クラスについて主要な意味概念は、名前が名前クラスに属することである。名前クラスは、生成規則 nameClass にマッチする要素である。名前は，**項 2** で定義されている。名前は、名前空間 URI と局所名から成る。

次に示す記法を用いる。

n

名前を値とする変数

nc

名前クラスを値とする変数

n in *nc*

n が名前クラス *nc* に属することを意味する

最初の公理、"anyName 1"を次に示す。

(anyName 1)　*n* in <anyName/>

この公理は，どんな名前 *n* についても，*n* は名前クラス<anyName/>に属することを定める。この公理に現れる変数には、全称記号が暗黙に指定されていることに注意。この暗黙の指定があるため、この公理はどんな名前 *n* についても適用できる。

最初の推論規則も同程度に簡単である。

$$\text{(anyName 2)}\quad \frac{\text{not}(n \text{ in } nc)}{n \text{ in } \langle\text{anyName}\rangle \langle\text{except}\rangle\ nc\ \langle/\text{except}\rangle \langle/\text{anyName}\rangle}$$

この推論規則は，どんな名前 *n* 及び名前クラス *nc* に対しても，もし *n* が *nc* に属さないのなら，*n* は <anyName> <except> *nc* </except> </anyName>に属することを示す。言い換えれば，<anyName> <except> *nc* </except> </anyName> は *nc* にマッチしないどんな名前にもマッチする。

次に示す補助的な記法が必要になる。

ln

局所名を値とする変数。局所名は文字列であり、[XML Namespaces]の NCName 生成規則にマッチするもの（すなわちコロンを含まない名前）である。

u

URI を値とする変数

name(*u*, *ln*)

URI*u* と局所名 *ln* をもつ名前を構築する

名前クラスについての他の公理及び推論規則を次に示す。

(nsName 1)　name(*u*, *ln*) in <nsName ns="*u*"/>

(nsName 2)　$\dfrac{\text{not(name(} u, ln \text{) in } nc)}{\text{name(} u, ln \text{) in <nsName ns="} u \text{"> <except> } nc \text{ </except> </nsName>}}$

(name)　name(*u*, *ln*) in <name ns="*u*"> *ln* </name>

(name choice 1)　$\dfrac{n \text{ in } nc_1}{n \text{ in <choice> } nc_1\ nc_2 \text{ </choice>}}$

(name choice 2)　$\dfrac{n \text{ in } nc_2}{n \text{ in <choice> } nc_1\ nc_2 \text{ </choice>}}$

6.1. 長さクラス

長さクラスについて主要な意味概念は、長さが長さクラスに属することである。長さクラスは、生成規則 lengthClass にマッチする要素である。長さは，**項 2** で定義されている。

次に示す記法を用いる。

l

長さを値とする変数

lc

長さクラスを値とする変数

l in *lc*

l が長さクラス *lc* に属することを意味する

最初の公理、"anyLength 1"を次に示す。

(anyLength 1)　*l* in <anyLength/>

この公理は，どんな長さ *l* についても，*l* は長さクラス<anyLength/>に属することを定める。この公理に現れる変数には、全称記号が暗黙に指定されていることに注意。この暗黙の指定があるため、この公理はどんな名前 *l* についても適用できる。

最初の推論規則も同程度に簡単である。

$$\text{(anyLength 2)}\quad \frac{\text{not}(l \text{ in } lc)}{L \text{ in } \langle\text{anyLength}\rangle\ \langle\text{except}\rangle\ lc\ \langle/\text{except}\rangle\ \langle/\text{anyLength}\rangle}$$

この推論規則は，どんな長さ *l* 及び長さクラス *lc* に対しても，もし *l* が *lc* に属さないのなら， *l* は <anyLength> <except> *lc* </except> </anyLength>に属することを示す。言い換えれば， <anyLength> <except> *lc* </except> </anyLength> は *lc* にマッチしないどんな長さにもマッチする。

長さクラスについての他の公理及び推論規則を次に示す。

$$\text{(length)}\quad l \text{ in } \langle\text{length}\rangle\ l\ \langle/\text{length}\rangle$$

$$\text{(length choice 1)}\quad \frac{l \text{ in } lc_1}{l \text{ in } \langle\text{choice}\rangle\ lc_1\ lc_2\ \langle/\text{choice}\rangle}$$

$$\text{(length choice 2)}\quad \frac{l \text{ in } lc_2}{l \text{ in } \langle\text{choice}\rangle\ lc_1\ lc_2\ \langle/\text{choice}\rangle}$$

6.2. パターン

パターンのための公理と推論規則では、次の記法を用いる。

cx

文脈（**項 2** で定義したもの）を値とする変数

a

属性の集合を値とする変数。一つの属性しか含まない集合は，その属性と同一視する。

m

要素及び文字列ビット列からなる列を値とする変数。一つの要素又は文字列ビット列しか含まない列は，その要素又は文字列ビット列と同一視する。*m* の値である列は、文字列ビット列が連続したものを含んでもよく，空の文字列ビット列を含んでもよい。したがって、*m* の値である列であって、要素の子（複数）としては出現できないものがある。

p

パターン（パターン生成規則にマッチする要素）を値とする変数

en

エンコードを値とする変数

cx |– a; m =~ p –|en,
文脈 cx において、属性(複数)a 並びに要素及び文字列ビット列からなる列 m がエンコード en においてパターン p にマッチすることを意味する。

6.2.1. choice パターン

choice パターンの意味を次のとおりに定める。

$$\text{(choice 1)}\quad \frac{cx \text{ |– } a;\ m \text{ =~ } p_1 \text{ –|}en,}{cx \text{ |– } a;\ m \text{ =~ } \langle\text{choice}\rangle\ p_1\ p_2\ \langle/\text{choice}\rangle \text{ –|}en,}$$

$$\text{(choice 2)}\quad \frac{cx \text{ |– } a;\ m \text{ =~ } p_2 \text{ –|}en,}{cx \text{ |– } a;\ m \text{ =~ } \langle\text{choice}\rangle\ p_1\ p_2\ \langle/\text{choice}\rangle \text{ –|}en,}$$

6.2.2. group パターン

次に示す補助的な記法を用いる。

m_1, m_2

列 m_1 と列 m_2 を連結したものを表す。

a_1 + a_2

a_1 と a_2 の和集合を表す。

group パターンの意味を次のとおりに定める。

$$\text{(group)}\quad \frac{cx \text{ |– } a_1;\ m_1 \text{ =~ } p_1 \text{ –|}en, \qquad cx \text{ |– } a_2;\ m_2 \text{ =~ } p_2 \text{ –|}en,}{cx \text{ |– } a_1 + a_2;\ m_1, m_2 \text{ =~ } \langle\text{group}\rangle\ p_1\ p_2\ \langle/\text{group}\rangle \text{ –|}en,}$$

注意

項 7.3 に示す制限は、推論規則の結論において構築される属性集合に、同一の名前をもつ複数の属性がないことを保証する。

6.2.3. empty パターン

次に示す補助的な記法を用いる。

( )

空の列を表す。

{ }

空の集合を表す。

{*}

解決されたエンコードを表す。

empty パターンの意味を次のとおりに定める。

(empty)　$cx \vdash \{\}; (\,) =^{\sim} \langle \mathtt{empty}/\rangle \dashv\!\vdash \{en\},$

6.2.4. text パターン

次に示す補助的な記法を用いる。

s

文字列ビット列を値とする変数

text パターンの意味を次のとおりに定める。

(text 1)　$cx \vdash \{\}; (\,) =^{\sim} \langle \mathtt{text}/\rangle$

(text 2)　$$\frac{cx \vdash \{\}; m =^{\sim} \langle \mathtt{text}/\rangle}{cx \vdash \{\}; m, s =^{\sim} \langle \mathtt{text}/\rangle}$$

(text)　$s =^{\sim} \langle \mathtt{text}/\rangle \dashv\!\vdash en,$

この規則は，text は 0 個以上の文字列ビット列にマッチすることを意味している。

6.2.5. oneOrMore パターン

次に示す補助的な記法を用いる。

disjoint(a_1, a_2)

a_1 にある属性の名前であって，しかも a_2 にある属性の名前であるものは存在しないことを意味している。

oneOrMore パターンの意味を次のとおりに定める。

(oneOrMore 1)
$$\frac{cx \vdash a;\ m =\tilde{}\ p\ \dashv en,}{cx \vdash a;\ m =\tilde{}\ \langle \mathtt{oneOrMore} \rangle\ p\ \langle /\mathtt{oneOrMore} \rangle\ \dashv en,}$$

(oneOrMore 2)
$$\frac{cx \vdash a_1;\ m_1 =\tilde{}\ p \dashv en,\quad cx \vdash a_2;\ m_2 =\tilde{}\ \langle \mathtt{oneOrMore} \rangle\ p\ \langle /\mathtt{oneOrMore} \rangle\ \dashv en,\ \ \mathrm{disjoint}(a_1, a_2)}{cx \vdash a_1 + a_2;\ m_1, m_2 =\tilde{}\ \langle \mathtt{oneOrMore} \rangle\ p\ \langle /\mathtt{oneOrMore} \rangle\ \dashv en,}$$

6.2.6. interleave パターン

次に示す補助的な記法を用いる。

m_1 interleaves m_2; m_3

m_1 は、 m_2 及び m_3 のインタリーブであることを示す。

インタリーブの意味を次の推論規則によって定める。

(interleaves 1)　( ) interleaves ( ); ( )

(interleaves 2)
$$\frac{m_1 \text{ interleaves } m_2;\ m_3}{m_4,\ m_1 \text{ interleaves } m_4,\ m_2;\ m_3}$$

(interleaves 3)
$$\frac{m_1 \text{ interleaves } m_2;\ m_3}{m_4,\ m_1 \text{ interleaves } m_2;\ m_4,\ m_3}$$

例えば，〈a/〉〈a/〉と 〈b/〉 のインタリーブは 〈a/〉〈a/〉〈b/〉，〈a/〉〈b/〉〈a/〉及び 〈b/〉〈a/〉〈a/〉 である。

例えば，ビット列の列 ″A″ ″A″ とビット列の列 ″B″ のインターリーブは

ビット列の列 ″A″ ″A″ ″B″，″A″ ″B″ ″A″ 及び ″B″ ″A″ ″A″ である。*****

interleave パターンの意味を次のとおりに定める。

(interleave)
$$\frac{cx \vdash a_1;\ m_1 =\tilde{}\ p_1\ \dashv en,\quad cx \vdash a_2;\ m_2 =\tilde{}\ p_2\ \dashv en,\quad m_3 \text{ interleaves } m_1;\ m_2}{cx \vdash a_1 + a_2;\ m_3 =\tilde{}\ \langle \mathtt{interleave} \rangle\ p_1\ p_2\ \langle /\mathtt{interleave} \rangle\ \dashv en,}$$

注意

項 7.3 にある制限は，推論規則の結論において構築される属性集合は、同一の名前をもつ複数の属性をもたないことを保証する。

6.2.7. element 及び attribute パターン

属性の値は，常に単一の文字列である。文字列が空であってもよい。したがって、空の列は属性の値とはなり得ない。一方，要素の子（複数）は空の列であってもよいが、空の文字列であることはない。検証が要素と属性を統一的に扱うことを保証するため，マッチの変種として *弱い*マッチを導入する。弱いマッチ は、属性の値をパターンにマッチさせるとき、属性(複数)並びに要素の子(複数)をパターンにマッチさせるときに用いる。弱いマッチを定義するために，次の記法を用いる。

""

空の文字列を表す。

ws

空列を値とする、又は空白のみから構成される文字列を値とする変数

cx |– *a*; *m* $=\tilde{}_{\text{weak}}$ *p*

文脈 *cx* において、属性 *a* 並びに要素及び文字からなる列 *m* とが、パターン *p* と弱いマッチをすることを意味する。

弱いマッチの意味を次のとおりに定める。

$$\text{(weak match 1)}\quad \frac{cx \vdash a;\ m =\tilde{}\ p}{cx \vdash a;\ m =\tilde{}_{\text{weak}}\ p}$$

$$\text{(weak match 2)}\quad \frac{cx \vdash a;\ (\,) =\tilde{}\ p}{cx \vdash a;\ ws =\tilde{}_{\text{weak}}\ p}$$

$$\text{(weak match 3)}\quad \frac{cx \vdash a;\ "" =\tilde{}\ p}{cx \vdash a;\ (\,) =\tilde{}_{\text{weak}}\ p}$$

次に示す補助的な記法を用いる。

attribute(*n*, *s*)

名前 *n* と値 *s* をもつ属性を構築する。

element(*n*, *cx*, *a*, *m*)

名前が *n* で、文脈が *cx* で、属性(複数)が *a* で、子(複数)が混在列 *m* である要素を構築する。

okAsChildren(*m*)

混在列 *m* が、要素の子(複数)として出現できるものであることを示す。すなわち、空の文字列が現れることはなく，二つの文字列が連続として現れることもない。

deref(*ln*) = <element> *nc p* </element>

文法が <define name="*ln*"> <element> *nc p* </element> </define>を含むことを意味する。

attribute パターンの意味を次のとおりに定める。

$$\text{(attribute)}\quad \frac{cx \vdash \{\};\ s =\tilde{}_{\text{weak}}\ p \qquad n \text{ in } nc}{cx \vdash \text{attribute}(\ n,\ s\ );(\ ) =\tilde{}\ \langle\text{attribute}\rangle\ nc\ p\ \langle/\text{attribute}\rangle}$$

element パターンの意味を次のとおりに定める。

$$\text{(element)}\quad \frac{cx_1 \vdash a;\ m =\tilde{}_{\text{weak}}\ p \quad n \text{ in } nc \quad \text{okAsChildren}(m) \quad \text{deref}(ln) = \langle\text{element}\rangle\ nc\ p\ \langle/\text{element}\rangle}{cx_2 \vdash \{\};\ ws_1, \text{element}(\ n,\ cx_1,\ a,\ m\ ),\ ws_2 =\tilde{}\ \langle\text{ref name}=''ln''/\rangle}$$

6.2.7. bit 及び byte パターン

次に示す補助的な記法を用いる。

concatenate(*m*)

列 *m* に含まれるビット列を順に連結したものを表す。なお、空の列を連結したものは空のビット列である。

bitlen(*s*)

ビット列 *s* のビット長を表す。

bytelen (*s*)

ビット列 *s* のバイト長を表す。

deref(*ln*) = *p*

文法が <define name="*ln*"> *p* </define>を含むことを意味する。

bit 及び byte パターンの意味を次のとおりに定める。

$$\text{(bit1)}\quad \frac{m =\tilde{}\ p \dashv\vdash en,\quad \text{bitlen}(\ \text{concatenate}(\ m\ )\ )\ \text{in } lc \quad \text{deref}(ln) = \langle\text{bit}\rangle\ lc\ p\ \langle/\text{bit}\rangle}{\text{concatenate}(\ m\ ) =\tilde{}\ \langle\text{ref name}=''ln''/\rangle\ \dashv\vdash en,}$$

$$\text{(byte1)}\quad \frac{m =\tilde{}\ p \dashv\vdash en,\quad \text{bytelen}(\ \text{concatenate}(\ m\ )\ )\ \text{in } lc \quad \text{deref}(ln) = \langle\text{byte}\rangle\ lc\ p\ \langle/\text{byte}\rangle}{\text{concatenate}(\ m\ ) =\tilde{}\ \langle\text{ref name}=''ln''/\rangle\ \dashv\vdash en,}$$

$$\text{(bit2)}\quad \frac{m =\tilde{}\ p \dashv\vdash en,\quad \text{bitlen}(\ \text{concatenate}(\ m\ )\ )\ \text{in } lc \quad \text{deref}(ln) = \langle\text{bit encode}=''en''\rangle\ lc\ p\ \langle/\text{bit}\rangle}{\text{concatenate}(\ m\ ) =\tilde{}\ \langle\text{ref name}=''ln''/\rangle\ \dashv\vdash\{*\},}$$

$$\text{(byte2)}\quad \frac{m =\tilde{}\ p \dashv\vdash en,\quad \text{bytelen}(\ \text{concatenate}(\ m\ )\ )\ \text{in } lc \quad \text{deref}(ln) = \langle\text{byte encode}=''en''\rangle\ lc\ p\ \langle/\text{byte}\rangle}{}$$

concatenate(*m*) =~ <ref name="*ln*"/> -|{*},

6.2.8. data 及び value パターン

RELAX NG UMS は、データ型についてはデータ型ライブラリに委ねている。データ型ライブラリは URI によって特定する。あるデータ型ライブラリに属するデータ型は NCName によって特定する。データ型ライブラリは二つの機能をもつ。

- ある文字列ビット列が、あるデータ型において許される表現であるかどうかを決定することができる。この機能は，0 個以上のパラメタからなるリストを受け取る。例えば，文字列データ型は、文字列の長さを指定するパラメタをもつかもしれない。データ型ライブラリは，各データ型に適用できるパラメタが何かを定める。
- 二つの文字列ビット列が、あるデータ型の同じ値を表すかどうかを決定することができる。この機能は，パラメタをもたない。

どちらの機能も，文字列が現れる文脈を用いるかもしれない。例えば，QName を表すデータ型は名前空間マップを用いる。どちらの機能も，エンコードを用いる。

次に示す補助的な記法を用いる。

datatypeAllows(*u*, *ln*, *params*, *s*, *cx en*)

URI*u* が示すデータ型ライブラリで、文脈 *cx* エンコード *en* において解釈される文字列ビット列 *s* は、パラメタ *params* をもつデータ型 *ln* の合法的な値であることを意味する。

datatypeEqual(*u*, *ln*, s_1, cx_1 en_1, s_2, cx_2 en_2)

URI*u* が示すデータ型ライブラリで、文脈 cx_1 エンコード en_1 において解釈される文字列ビット列 s_1 と文脈 cx_2 エンコード en_2 において解釈される文字列ビット列 s_2 は、データ型 *ln* の同じ値をもつことを意味する。

params

パラメタの列を値とする変数

[*cx*]

パターンを表す開始タグの中では、そのパターン要素の文脈のことを意味する。

context(*u*, *cx*)

cx と同一であるが，デフォルトの名前空間が *u* である点だけが異なる文脈を構築する。もし、 *u* が空文字列であれば、構築された文脈はデフォルト名前空間をもたない。

datatypeEqual 関数は、反射的・推移的・対称的でなければならない。すなわち、次の推論規則が成立しなくてはならない。

$$\text{(datatypeEqual reflexive)}\quad \frac{\text{datatypeAllows}(u, ln, params, s, cx\ en)}{\text{datatypeEqual}(u, ln, s, cx\ en, s, cx\ en)}$$

(datatypeEqual transitive)
$$\frac{\text{datatypeEqual}(u, ln, s_1, cx_1\ en_1, s_2, cx_2\ en_2)\ \ \text{datatypeEqual}(u, ln, s_2, cx_2\ en_2, s_3, cx_3\ en_3)}{\text{datatypeEqual}(u, ln, s_1, cx_1\ en_1, s_3, cx_3\ en_3)}$$

(datatypeEqual symmetric)
$$\frac{\text{datatypeEqual}(u, ln, s_1, cx_1\ en_1, s_2, cx_2\ en_2)}{\text{datatypeEqual}(u, ln, s_2, cx_2\ en_2, s_1, cx_1\ en_1)}$$

data パターン及び value パターンの意味を次のとおりに定める。

(value)
$$\frac{\text{datatypeEqual}(u_1, ln, s_1, cx_1\ en, s_2, \text{context}(\ u_2, cx_2\ )\ en)}{cx_1 \mid\!\!- \{\ \};\ s_1 \mathrel{=\!\!\sim} \langle\text{value datatypeLibrary}=\!''u_1''\ \text{type}=\!''ln''\ \text{ns}=\!''u_2''\ [cx_2]\rangle\ s_2\ \langle/\text{value}\rangle\ -\!\mid\!en,}$$

(data 1)
$$\frac{\text{datatypeAllows}(u, ln, params, s, cx\ en)}{cx \mid\!\!- \{\ \};\ s \mathrel{=\!\!\sim} \langle\text{data datatypeLibrary}=\!''u''\ \text{type}=\!''ln''\rangle\ params\ \langle/\text{data}\rangle\ -\!\mid\!en,}$$

(data 2)
$$\frac{\text{datatypeAllows}(u, ln, params, s, cx\ en) \qquad \text{not}(cx \mid\!\!- a;\ s \mathrel{=\!\!\sim} p)}{cx \mid\!\!- \{\ \};\ s \mathrel{=\!\!\sim} \langle\text{data datatypeLibrary}=\!''u''\ \text{type}=\!''ln''\rangle\ params\ \langle\text{except}\rangle\ p\ \langle/\text{except}\rangle\ \langle/\text{data}\rangle\ -\!\mid\!en,}$$

注意

データ型ライブラリは異なるエンコードのビット列を比較できる。この機能は妥当性の検証で使用されない。

6.2.9. 組み込みデータ型ライブラリ

空の URI は、特別の組み込みデータ型ライブラリを特定する。このライブラリは，一つのエンコード型 txt，二つのデータ型 string と token を提供する。どちらのデータ型もパラメタをもつことはできない。

$s_1 = s_2$

s_1 と s_2 が同一であることを意味する。

normalizeWhiteSpace(s)

ビット列 s をエンコード型 txt で解釈した文字列の先頭と末尾にある空白文字を取り除き、それ以外の空白文字の列を単一のスペース文字に置き換えたものを返す。

二つの組み込みデータ型の意味を次のとおりに定める。

(string allows) datatypeAllows("", "string", ( ), s, cx "txt")

(string equal) datatypeEqual("", "string", s, cx_1 "txt", s, cx_2 "txt")

(token allows)　datatypeAllows("", "token", ( ), *s*, *cx* "txt")

(token equal)
$$\frac{\text{normalizeWhiteSpace}(\ s_1\ ) = \text{normalizeWhiteSpace}(\ s_2\ )}{\text{datatypeEqual}(\text{"}\text{"}, \text{"token"}, s_1, cx_1\ \text{"txt"}, s_2, cx_2\ \text{"txt"})}$$

6.2.10. list パターン

次に示す補助的な記法を用いる。

split(*s*, *en*)

空白によって区切られたトークンの一つ一つに対して一つの *s* 文字列をもつ列を返す。返された列に含まれる各文字列は、空ではなく空白を含まない。

=== RELAX NG 訳文は誤記か？ ===

ビット列 *s* をエンコード型 *en* で文字列として解釈し、空白によって区切られたトークンの一つ一つに対して一つのビット列をもつ列を返す。返された列に含まれる各ビット列をエンコード型 *en* で解釈した文字列は、空ではなく空白を含まない。

separate(*s*, *c*, *en*)

ビット列 *s* をエンコード型 *en* で文字列として解釈し、文字 *c* によって区切られたトークンの一つ一つに対して一つのビット列をもつ列を返す。返された列は空ではなく、各ビット列をエンコード型 *en* で解釈した文字列は、文字 *c* を含まない。

例えば separate("A,B,C", ",") は、列 "A" "B" "C" を返す。

delimit(*s*, *c*, *en*)

ビット列 *s* をエンコード型 *en* で文字列として解釈し、引き続く文字 *c* によって区切られたトークンの一つ一つに対して一つのビット列をもつ列を返す。返された列は空ではなく、各ビット列をエンコード型 *en* で解釈した文字列は、文字 *c* を含まない。

例えば delimit("A,B,C", ",") は、列 "A" "B" を返す。

list パターンの意味を次のとおりに定める。

(list)
$$\frac{cx\ |\!-\ \{\};\ \text{split}(\ s, en\ ) =\!\sim\ p\ -|en,}{cx\ |\!-\ \{\};\ s =\!\sim\ \langle\text{list}\rangle\ p\ \langle/\text{list}\rangle\ -|en,}$$

(list separate)
$$\frac{\text{separate}(\ s, c, en\ ) =\!\sim\ p\ -|en,}{s =\!\sim\ \langle\text{list separator=}\text{"}c\text{"}\rangle\ p\ \langle/\text{list}\rangle\ -|en,}$$

(list delimit)
$$\frac{\text{delimit}(\ s, c, en\ ) =\!\sim\ p\ -|en,}{s =\!\sim\ \langle\text{list delimiter=}\text{"}c\text{"}\rangle\ p\ \langle/\text{list}\rangle\ -|en,}$$

注意

直前に示した推論規則において、パターンと照合される列は連続するいくつかの文字列ビット列を含んでもよいことに注意。

6.3. 妥当性

要素がスキーマに照らして妥当であるのはどんな場合かを定義する。次に示す補助的な記法を用いる。

e

要素を値とする変数

valid(*e*)

要素 *e* が文法に照らして妥当であることを意味する

start() = *p*

文法が 〈start〉 *p* 〈/start〉 を含むことを意味する

要素が妥当であるのは，この要素に空の属性集合を組み合わせたものが、文法の start パターンにマッチするときである。

$$\frac{\text{start}() = p \quad cx \vdash \{\}; e =^{\sim} p \dashv\vdash \{*\},}{\text{valid}(e)}\ (\text{valid})$$

RELAX NG は何故空の属性集合とマッチさせる？？

6.4. 例

e_0 の値が

element(name("", "foo"), cx_0, { }, *m*)

であると仮定する。ここで，*m* は

e_1, e_2

であり e_1 は

element(name("http://www.example.com/n1", "bar1"), cx_1, { }, ( ))
element(name("http://www.example.com/n2", "bar2"), cx_2, { }, ( ))

である。

cx_0, cx_1 及び cx_2 が適切に定義されているとすれば，これらは 項 2.1 に示す文書を表現する。

スキーマ 項 5.1 に照らして e_0 が妥当であることをいかにして示すかを説明する。このスキーマは，次の命題（複数）と同等である。

start() = <ref name="foo"/>
deref("foo.element") = <element> <name ns=""> "foo" </name> <group> <ref name="bar1"/> <ref name="bar2"/> </group> </element>
deref("bar1.element") = <element> <name ns="http://www.example.com/n1"> "bar1" </name> <empty/> </element>
deref("bar2.element") = <element> <name ns="http://www.example.com/n2"> "bar2" </name> <empty/> </element>

名前クラス nc_1 が

<name ns="http://www.example.com/n1"> "bar1" </name>
<name ns="http://www.example.com/n2"> "bar2" </name>

であるとする。

このとき、項 6.1 の推論規則(name)によって,

name("http://www.example.com/n1", "bar1") in nc_1
name("http://www.example.com/n2", "bar2") in nc_2

が得られる。

項 6.2.3 の推論規則(empty) によって、

cx_1 |– { }; ( ) =~ <empty/>
cx_2 |– { }; ( ) =~ <empty/>

が得られる。

cx_0 |– { }; e_1 =~ <ref name="bar1"/>

が得られる。

任意の文脈が利用できるので, cx_0 を選んでいることに注意する。

cx_0 |– { }; e_2 =~ <ref name="bar2"/>

が得られる。

cx_0 |– { }; e_1, e_2 =~ <group> <ref name="bar1"/> <ref name="bar2"/> </group>

が得られる。

項 6.2.7 の推論規則(element)によって、

cx_3 |– { }; element(name("", "foo"), cx_0, { }, m) =~ <ref name="foo"/>

が得られる。ここで, cx_3 は任意の文脈である。

valid(e_0)

が得られる。

ビット列 s をエンコード型 en で文字列として解釈し、

s をエンコード型 en で文字列として解釈すると値が "ABC" となるビット列とする。これは 項 2.1 に示す文書を表現する。

スキーマ 項 5.1 に照らして s が妥当であることをいかにして示すかを説明する。このスキーマは，次の命題(複数)と同等である。

start() = <ref name="byte1"/>
deref("byte1") = <byte encode="txt"> <anyLength/> <group> <ref name="byte2"/> <ref name="byte3"/> </group> </byte>
deref("byte2") = <byte> <length>1</length> <text/> </byte>
deref("byte3") = <byte> <length>2</length> <text/> </byte>

長さクラス lc_1 lc_2 が

<length> "1" </length>
<length> "2" </length>

であるとする。

このとき、項 6.1 の推論規則(length)によって，

length("A") in lc_1
length("BC") in lc_2

が得られる。

項 6.2.4 の推論規則(text) によって、

"A" =~ <text/> |-txt,
"BC" =~ <text/> |-txt,

が得られる。

s_1 =~ <ref name="byte2"/> |-txt,

が得られる。

s_2 =~ <ref name="byte3"/> |-txt,

が得られる。

s_1, s_2 =~ <group> <ref name="byte2"/> <ref name="byte3"/> </group> |-txt,

が得られる。

項 6.2.7 の推論規則(byte)によって、

length(concatenate(s_1, s_2)) in <anyLength/>

concatenate(s_1, s_2) =~ <ref name="byte"/> |-{*},

が得られる。

valid(e_0)

が得られる。

7. 制限

次に示す制約は文法が項 5 で規定した単純な形式に変換された後にすべて確認される。これらの制限は、利用者の誤りの検出と実装の簡略化を目的とする。

7.1. 文脈上の制限

この節では、スキーマにおいて要素が出現できる位置についての制限を、祖先要素の名前に基づいて規定する。これらの制限を規定するために*禁止された経路*の概念を使用する。経路は /又は// で区切られた NCName の並びとなる。

- ある要素が経路 *x* にマッチするのは，その要素のローカル名が *x* のときに限る。ここで *x* は NCName とする。
- ある要素が経路 *x*/*p* にマッチするのは，この要素の局所名が *x* であり、この要素のある子が *p* にマッチするときに限る。ここで *x* は NCName で、*p* は経路とする。
- ある要素が経路 *x*//*p* にマッチするのは，この要素の局所名が *x* であり、この要素のある子孫要素が *p* にマッチするときに限る。ここで *x* は NCName で、 *p* は経路とする。

たとえば，要素

```
<foo>
  <bar>
    <baz/>
  </bar>
</foo>
```

は，経路 foo, foo/bar, foo//bar, foo//baz, foo/bar/baz, foo/bar//baz 及び foo//bar/baz にマッチするが，foo/baz 又は foobar にはマッチしない。

単純な形式に変換された後の正しい RELAX NG UMS スキーマは，禁止された経路にマッチする要素を含んではならない。

7.1.1. attribute パターン

次に示す経路は禁止されている。

- attribute//ref
- attribute//attribute

7.1.2. oneOrMore パターン

次に示す経路は禁止されている。

- oneOrMore//group//attribute
- oneOrMore//interleave//attribute

7.1.3. list パターン

次に示す経路は禁止されている。

- list//list # 異なる区切りを指定すればネストできる
- list//ref # 許容される
- list//attribute
- list//text # UMS の text は列ではないビット列にマッチするためこの制約への必然性が低いが、RELAX NG ユーザの混乱を避けるため仕様をあわせる
- list//interleave # 許容される

7.1.4. data 中の except パターン

次に示す経路は禁止されている。

- data/except//attribute
- data/except//ref
- data/except//text
- data/except//list
- data/except//group
- data/except//interleave
- data/except//oneOrMore
- data/except//empty

注意

これは data を親にもつ except 要素は data, value, 及び choice 要素だけを含むことができることを意味する。

7.1.5. start 要素

データモデルの「ルートノード」に何を許容するかのための制約。RELAX NG は、XML データを対象とするため列ではない要素のみを許容する。UMS においては列ではないビット列のみを許容する。

次に示す経路は禁止されている。

- start//attribute
- start//data
- start//value
- start//text
- start//list # UMS においても、エンコードが未確定な状態の list は許容されない
- start//group
- start//interleave
- start//oneOrMore
- start//empty

注意

これは start 要素が ref, 及び choice 要素だけを含むことができることを意味する。

7.2. 文字列の並び

RELAX NG UMS は次のようなパターンを許可しない。

```
<element name="foo">
  <group>
    <data type="int"/>
    <element name="bar">
      <empty/>
    </element>
  </group>
</element>

<byte length="5">
  <group>
    <data type="int"/>
    <byte length="1">
      <empty/>
    </byte>
  </group>
</byte>
```

次のようなパターンもまた許可しない。

```
<element name="foo">
  <group>
    <data type="int"/>
    <text/>
  </group>
</element>

<byte length="5">
  <group>
    <data type="int"/>
    <text/>
  </group>
</element>
```

一般的に説明する。もし byte 又は bit 要素又は属性の内容のためのパターンが

- 子にマッチすることができるパターン (byte, bit, element, data, value, list 又は text パターン)及び
- 一つの文字列にマッチするパターン (data, 又は value 又は list パターン)

含んでいるなら, これら二つのパターンが同時に適用されることはない。

この規則は list パターン中に出現するパターンには適用しない。

この規則を形式化するために, 内容種別の概念を用いる。byte 又は bit 要素の内容として許可されるパターンは emtpy, complex, simple の三つの内容種別のうち一つをもつ。次に示す記法を用いる。

empty( )

empty 内容種別を返す

complex( )

complex 内容種別を返す

simple( )

simple 内容種別を返す

ct

内容種別を値とする変数

内容種別を持つ = byte, bit, element の内容として許容されるといういみ。この節を最後まで読むと byte, bit, element の内容のためのパターンは empty(), complex(), simple() のみであり、それ以外は許されないことが分かる。

groupable(ct_1, ct_2)

内容種別 ct_1 と ct_2 がグループ化可能であることを示す。

empty 内容種別はどの内容種別ともグループ化可能である。さらに, complex 内容種別は complex 内容種別とグループ化可能である。次に示す規則はこれを形式化している。

(group empty 1) groupable(empty( ), *ct*)

(group empty 2) groupable(*ct*, empty( ))

(group complex) groupable(complex( ), complex( ))

いくつかのパターンは内容種別をもつ。次に示す補助的な記法を用いる。

$p :_c ct$

パターン p は内容種別 ct をもつことを示す

max(ct_1, ct_2)

ct_1 と ct_2 の大きいほうを返すここで、内容種別は empty( ), complex( ), simple( )empty( ), complex( ), simple( ) の順に大きいとする。

empty(), complex(), simple() を繰り返すのは Web から入手した RELAX NG 仕様書（3 December 2001）日本語版の誤記と思われる。

注意: empty(), complex(), simple() の順に大きくなる。語幹とは順序が異なる。ここで大小関係は、他のパターンとの組み合わせの困難さを元に設定されている。empty() が組み合わせやすく、simple() がもっとも困難である。

次に示す規則は，パターンが内容種別をもつのはどんな場合か，その場合の内容種別は何かを定義している。

(value) <value datatypeLibrary="u_1" type="*ln*" ns="u_2"> *s* </value> $:_c$ simple( )

(data 1) <data datatypeLibrary="*u*" type="*ln*"> *params* </data> $:_c$ simple( )

(data 2) $\dfrac{}{p :_c ct}$

<data datatypeLibrary="*u*" type="*ln*"> *params* <except> *p* </except> </data> :$_c$ simple( )

(list) <list> *p* </list> :$_c$ simple( ) complex( )

(list separate) <list separator="*c*"> *p* </list> :$_c$ complex( )

(list delimit) <list delimiter="*c*"> *p* </list> :$_c$ complex( )

(text) <text/> :$_c$ complex( ) simple( )

(ref) <ref name="*ln*"/> :$_c$ complex( )

(empty) <empty/> :$_c$ empty( )

(attribute)
$$\frac{p :_c ct}{\text{<attribute> } nc\ p \text{ </attribute>} :_c \text{ empty( )}}$$

(group)
$$\frac{p_1 :_c ct_1 \quad p_2 :_c ct_2 \quad \text{groupable}(ct_1, ct_2)}{\text{<group> } p_1\ p_2 \text{ </group>} :_c \max(\ ct_1, ct_2\ )}$$

(interleave)
$$\frac{p_1 :_c ct_1 \quad p_2 :_c ct_2 \quad \text{groupable}(ct_1, ct_2)}{\text{<interleave> } p_1\ p_2 \text{ </interleave>} :_c \max(\ ct_1, ct_2\ )}$$

(oneOrMore)
$$\frac{p :_c ct \quad \text{groupable}(ct, ct)}{\text{<oneOrMore> } p \text{ </oneOrMore>} :_c ct}$$

(choice)
$$\frac{p_1 :_c ct_1 \quad p_2 :_c ct_2}{\text{<choice> } p_1\ p_2 \text{ </choice>} :_c \max(\ ct_1, ct_2\ )}$$

注意

[項 7.1.4](#) においていくつかの経路を禁止したため，規則（data2）の条件は冗長である。

制限を記述する準備ができた。次に示す記法を用いる。

incorrectSchema()

スキーマが正しくないことを示す

byte 又は bit 要素パターンの内容として出現するパターンはすべて内容種別を持たなければならない。

(element)
$$\frac{\text{deref}(ln) = \text{<element> } nc\ p \text{ </element>} \quad \text{not}(p :_c ct)}{}$$

incorrectSchema()

$$\text{(bit)}\quad \frac{\text{deref}(ln) = \langle\texttt{bit}\rangle\ lc\ p\ \langle/\texttt{bit}\rangle \quad \text{not}(p :_c ct)}{\text{incorrectSchema}()}$$

$$\text{(byte)}\quad \frac{\text{deref}(ln) = \langle\texttt{byte}\rangle\ nc\ p\ \langle/\texttt{byte}\rangle \quad \text{not}(p :_c ct)}{\text{incorrectSchema}()}$$

7.3. 属性の制限

属性の重複は許可しない。より正確には，パターン 〈group〉 *p1 p2* 〈/group〉 又は 〈interleave〉 *p1 p2* 〈/interleave〉 に対して， *p1* 中で出現する attribute パターンの名前クラスと *p2* 中で出現する attribute パターンの名前クラスの両方に属する名前があってはならない。パターン *p1* がパターン *p2* 中に出現するのは次の場合である

- *p1* が *p2* である場合，又は
- *p2* が choice, interleave, group 又は oneOrMore 要素であり *p1* が *p2* の片方又は両方の子の中に出現する場合

無限名前クラスに使われる属性は繰り返されなければならない。より正確には， anyName 又は nsName 子孫要素をもつ attribute 要素は oneOrMore 祖先要素を持たなければならない。

注意

この制限は否定下の閉包のために必要である。

7.4. interleave の制限

〈interleave〉 *p1 p2* 〈/interleave〉 パターンに対して，

- *p1* 中に出現する ref パターンに参照されている element パターンの名前クラスと *p2* 中に出現する ref パターンに参照されている element パターンの名前クラスとの両方に属する名前があってはならない。
- text パターンは *p1* と *p2* の両方に出現してはならない。

パターンが他のパターンの中に出現するのはどんな場合かは項 7.3 で定義した。

8. 適合性

適合する RELAX NG UMS 妥当性検証器は、XML 文書が正しい RELAX NG UMS スキーマであるかどうか決定できなければならない。適合する RELAX NG UMS 妥当性検証器は、XML 文書が正しい RELAX NG UMS スキーマに照らして妥当かどうか決定できなければならない。

前段落で述べた要求は、妥当性検証器が実装していないデータ型ライブラリをスキーマが使用していたときには適用されない。適合する RELAX NG UMS 妥当性検証器は 項 6.2.9 で規定した組込みデータ型ライブラリを実装することだけが要求される。 RELAX NG UMS に適合すると主張する妥当性検証器は実装するデータ型ライブラリを文書化するのが望ましい。スキーマが externalRef 又は include 要素を持っており、妥当性検証器

がその URI によって特定される資源を取り出せないとき，又は取り出した資源から要素を構築できないときも前段落の要求は適用されない。 RELAX NG UMS に適合すると主張する妥当性検証器については、 URI 参照の取扱い能力を文書化しておくことが望ましい。

附属書

A. RELAX NG UMS のための RELAX NG スキーマ

```
<grammar datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
  ns="http://relaxng.org/ns/structure/1.0"
  ns="http://ums.isas.jaxa.jp/0.4"
  xmlns="http://relaxng.org/ns/structure/1.0">

  <start>
    <ref name="pattern"/>
  </start>

  <define name="pattern">
    <choice>
      <element name="element">
        <choice>
          <attribute name="name">
            <data type="QName"/>
          </attribute>
          <ref name="open-name-class"/>
        </choice>
        <ref name="common-atts"/>
        <ref name="open-patterns"/>
      </element>
      <element name="attribute">
        <ref name="common-atts"/>
        <choice>
          <attribute name="name">
            <data type="QName"/>
          </attribute>
          <ref name="open-name-class"/>
        </choice>
        <interleave>
          <ref name="other"/>
          <optional>
            <ref name="pattern"/>
          </optional>
        </interleave>
      </element>
      <element name="byte">
        <optional>
          <attribute name="encode"/>
        </optional>
        <optional>
          <choice>
            <attribute name="length">
              <data type="int"/>
            </attribute>
            <ref name="open-length-class"/>
          </choice>
```

```
    </optional>
    <ref name="common-atts"/>
    <optional>
      <ref name="open-patterns"/>
    </optional>
  </element>
  <element name="bit">
    <optional>
      <attribute name="encode"/>
    </optional>
    <optional>
      <choice>
        <attribute name="length">
          <data type="int"/>
        </attribute>
        <ref name="open-length-class"/>
      </choice>
    </optional>
    <ref name="common-atts"/>
    <optional>
      <ref name="open-patterns"/>
    </optional>
  </element>
  <element name="group">
    <ref name="common-atts"/>
    <ref name="open-patterns"/>
  </element>
  <element name="interleave">
    <ref name="common-atts"/>
    <ref name="open-patterns"/>
  </element>
  <element name="choice">
    <ref name="common-atts"/>
    <ref name="open-patterns"/>
  </element>
  <element name="optional">
    <ref name="common-atts"/>
    <ref name="open-patterns"/>
  </element>
  <element name="zeroOrMore">
    <ref name="common-atts"/>
    <ref name="open-patterns"/>
  </element>
  <element name="oneOrMore">
    <ref name="common-atts"/>
    <ref name="open-patterns"/>
  </element>
  <element name="list">
    <optional>
      <choice>
        <attribute name="separator">
          <data type="string"/>
```

```
      </attribute>
      <attribute name="delimiter">
        <data type="string"/>
      </attribute>
    </choice>
  </optional>
  <ref name="common-atts"/>
  <ref name="open-patterns"/>
</element>
<element name="mixed">
  <ref name="common-atts"/>
  <ref name="open-patterns"/>
</element>
<element name="ref">
  <attribute name="name">
    <data type="NCName"/>
  </attribute>
  <ref name="common-atts"/>
  <ref name="other"/>
</element>
<element name="parentRef">
  <attribute name="name">
    <data type="NCName"/>
  </attribute>
  <ref name="common-atts"/>
  <ref name="other"/>
</element>
<element name="empty">
  <ref name="common-atts"/>
  <ref name="other"/>
</element>
<element name="text">
  <ref name="common-atts"/>
  <ref name="other"/>
</element>
<element name="value">
  <optional>
    <attribute name="type">
      <data type="NCName"/>
    </attribute>
  </optional>
  <ref name="common-atts"/>
  <text/>
</element>
<element name="data">
  <attribute name="type">
    <data type="NCName"/>
  </attribute>
  <ref name="common-atts"/>
  <interleave>
    <ref name="other"/>
    <group>
```

```
      <zeroOrMore>
        <element name="param">
          <attribute name="name">
            <data type="NCName"/>
          </attribute>
          <ref name="common-atts"/>
          <text/>
        </element>
      </zeroOrMore>
      <optional>
        <element name="except">
          <ref name="common-atts"/>
          <ref name="open-patterns"/>
        </element>
      </optional>
    </group>
  </interleave>
</element>
<element name="notAllowed">
  <ref name="common-atts"/>
  <ref name="other"/>
</element>
<element name="externalRef">
  <attribute name="href">
    <data type="anyURI"/>
  </attribute>
  <ref name="common-atts"/>
  <ref name="other"/>
</element>
<element name="grammar">
  <ref name="common-atts"/>
  <ref name="grammar-content"/>
</element>
</choice>
</define>

<define name="grammar-content">
  <interleave>
    <ref name="other"/>
    <zeroOrMore>
      <choice>
        <ref name="start-element"/>
        <ref name="define-element"/>
        <element name="div">
          <ref name="common-atts"/>
          <ref name="grammar-content"/>
        </element>
        <element name="include">
          <attribute name="href">
            <data type="anyURI"/>
          </attribute>
          <ref name="common-atts"/>
```

```
          <ref name="include-content"/>
        </element>
      </choice>
    </zeroOrMore>
  </interleave>
</define>

<define name="include-content">
  <interleave>
    <ref name="other"/>
    <zeroOrMore>
      <choice>
        <ref name="start-element"/>
        <ref name="define-element"/>
        <element name="div">
          <ref name="common-atts"/>
          <ref name="include-content"/>
        </element>
      </choice>
    </zeroOrMore>
  </interleave>
</define>

<define name="start-element">
  <element name="start">
    <ref name="combine-att"/>
    <ref name="common-atts"/>
    <ref name="open-pattern"/>
  </element>
</define>

<define name="define-element">
  <element name="define">
    <attribute name="name">
      <data type="NCName"/>
    </attribute>
    <ref name="combine-att"/>
    <ref name="common-atts"/>
    <ref name="open-patterns"/>
  </element>
</define>

<define name="combine-att">
  <optional>
    <attribute name="combine">
      <choice>
        <value>choice</value>
        <value>interleave</value>
      </choice>
    </attribute>
  </optional>
</define>
```

```
<define name="open-patterns">
  <interleave>
    <ref name="other"/>
    <oneOrMore>
      <ref name="pattern"/>
    </oneOrMore>
  </interleave>
</define>

<define name="open-pattern">
  <interleave>
    <ref name="other"/>
    <ref name="pattern"/>
  </interleave>
</define>

<define name="name-class">
  <choice>
    <element name="name">
      <ref name="common-atts"/>
      <data type="QName"/>
    </element>
    <element name="anyName">
      <ref name="common-atts"/>
      <ref name="except-name-class"/>
    </element>
    <element name="nsName">
      <ref name="common-atts"/>
      <ref name="except-name-class"/>
    </element>
    <element name="choice">
      <ref name="common-atts"/>
      <ref name="open-name-classes"/>
    </element>
  </choice>
</define>

<define name="except-name-class">
  <interleave>
    <ref name="other"/>
    <optional>
      <element name="except">
        <ref name="open-name-classes"/>
      </element>
    </optional>
  </interleave>
</define>

<define name="open-name-classes">
  <interleave>
    <ref name="other"/>
```

```
    <oneOrMore>
      <ref name="name-class"/>
    </oneOrMore>
  </interleave>
</define>

<define name="open-name-class">
  <interleave>
    <ref name="other"/>
    <ref name="name-class"/>
  </interleave>
</define>

<define name="length-class">
  <choice>
    <element name="length">
      <ref name="common-atts"/>
      <data type="int"/>
    </element>
    <element name="anyLength">
      <ref name="common-atts"/>
      <ref name="except-length-class"/>
    </element>
    <element name="choice">
      <ref name="common-atts"/>
      <ref name="open-length-classes"/>
    </element>
  </choice>
</define>

<define name="except-length-class">
  <interleave>
    <ref name="other"/>
    <optional>
      <element name="except">
        <ref name="open-length-classes"/>
      </element>
    </optional>
  </interleave>
</define>

<define name="open-length-classes">
  <interleave>
    <ref name="other"/>
    <oneOrMore>
      <ref name="length-class"/>
    </oneOrMore>
  </interleave>
</define>

<define name="open-length-class">
  <interleave>
```

```
    <ref name="other"/>
    <ref name="length-class"/>
  </interleave>
</define>

<define name="common-atts">
  <optional>
    <attribute name="ns"/>
  </optional>
  <optional>
    <attribute name="datatypeLibrary">
      <data type="anyURI"/>
    </attribute>
  </optional>
  <zeroOrMore>
    <attribute>
      <anyName>
        <except>
          <nsName/>
          <nsName ns=""/>
        </except>
      </anyName>
    </attribute>
  </zeroOrMore>
</define>

<define name="other">
  <zeroOrMore>
    <element>
      <anyName>
        <except>
          <nsName/>
        </except>
      </anyName>
      <zeroOrMore>
        <choice>
          <attribute>
            <anyName/>
          </attribute>
          <text/>
          <ref name="any"/>
        </choice>
      </zeroOrMore>
    </element>
  </zeroOrMore>
</define>

<define name="any">
  <element>
    <anyName/>
    <zeroOrMore>
      <choice>
```

```
          <attribute>
            <anyName/>
          </attribute>
          <text/>
          <ref name="any"/>
        </choice>
      </zeroOrMore>
    </element>
  </define>

</grammar>
```

B. 版 0.9 からの差分

(省略)

C. RELAX NG TC (参考)

この仕様書は、RELAX NG TC が準備し，公開を承認した。TC の現在のメンバを次に示す。

- Fabio Arciniegas
- James Clark
- Mike Fitzgerald
- 川口　耕介
- Josh Lubell
- 村田　真
- Norman Walsh
- David Webber

この仕様書を準備しているのは現在の所下記のメンバ

- 松崎　恵一

文献

引用規定

RFC 2396

T. Berners-Lee, R. Fielding, L. Masinter. *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax*. IETF (Internet Engineering Task Force). 1998.

RFC 2732

R. Hinden, B. Carpenter, L. Masinter. *RFC 2732: Format for Literal IPv6 Addresses in URL's*. IETF (Internet Engineering Task Force), 1999.

RFC 3023

村田　真, S. St.Laurent, D. Kohn. *RFC 3023: XML Media Types*. IETF (Internet Engineering Task Force), 2001.

XLink

Steve DeRose, Eve Maler and David Orchard, editors. *XML Linking Language (XLink) Version 1.0*. W3C (World Wide Web Consortium), 2001.

XML 1.0

Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen, Eve Maler, editors. *Extensible Markup Language (XML) 1.0 Second Edition*. W3C (World Wide Web Consortium), 2000.

XML Infoset

John Cowan, Richard Tobin, editors. *XML Information Set*. W3C (World Wide Web Consortium), 2001.

XML Namespaces

Tim Bray, Dave Hollander, and Andrew Layman, editors. *Namespaces in XML*. W3C (World Wide Web Consortium), 1999.

参考文献

RELAX

村田　真. *RELAX (Regular Language description for XML)*. INSTAC (Information Technology Research and Standardization Center), 2001.

TREX

James Clark. *TREX - Tree Regular Expressions for XML*. Thai Open Source Software Center, 2001.

Tutorial

James Clark, 村田　真, editors. *RELAX NG Tutorial*. OASIS, 2001.

W3C XML Schema Datatypes

Paul V. Biron, Ashok Malhotra, editors. *XML Schema Part 2: Datatypes*. W3C (World Wide Web Consortium), 2001.

XML Schema Formal

Allen Brown, Matthew Fuchs, Jonathan Robie, Philip Wadler, editors. *XML Schema: Formal Description*. W3C (World Wide Web Consortium), 2001.