

# Contents

	Page
Foreword.....	v
Introduction.....	vi
1 Scope.....	1
2 Normative references.....	1
3 Terms and definitions.....	2
4 Notation.....	2
4.1 EBNF.....	2
4.2 Inference rules.....	2
4.2.1 Variables.....	2
4.2.2 Propositions.....	2
4.2.3 Expressions.....	3
5 Data model.....	4
6 Full syntax.....	5
7 Simplification.....	6
7.1 General.....	6
7.2 Annotations.....	6
7.3 Whitespace.....	7
7.4 datatypeLibrary attribute.....	7
7.5 type attribute of value element.....	7
7.6 href attribute.....	7
7.7 externalRef element.....	7
7.8 include element.....	8
7.9 name attribute of element and attribute elements.....	8
7.10 ns attribute.....	8
7.11 QNames.....	8
7.12 div element.....	8
7.13 Number of child elements.....	8
7.14 mixed element.....	9
7.15 optional element.....	9
7.16 zeroOrMore element.....	9
7.17 Constraints.....	9
7.18 combine attribute.....	10
7.19 grammar element.....	10
7.20 define and ref elements.....	11
7.21 notAllowed element.....	11
7.22 empty element.....	11
8 Simple syntax.....	11
9 Semantics.....	12
9.1 Inference rules.....	12
9.2 Name classes.....	13
9.3 Patterns.....	13
9.3.1 choice pattern.....	13
9.3.2 group pattern.....	13
9.3.3 empty pattern.....	14
9.3.4 text pattern.....	14
9.3.5 oneOrMore pattern.....	14
9.3.6 interleave pattern.....	14
9.3.7 element and attribute pattern.....	15
9.3.8 data and value pattern.....	15
9.3.9 Built-in datatype library.....	16
9.3.10 list pattern.....	16
9.4 Validity.....	16

- 10 Restrictions..... 17
- 10.1 General..... 17
- 10.2 Prohibited paths..... 17
- 10.2.1 General..... 17
- 10.2.2 attribute pattern..... 17
- 10.2.3 oneOrMore pattern..... 17
- 10.2.4 list pattern..... 18
- 10.2.5 except element in data pattern..... 18
- 10.2.6 start element..... 18
- 10.3 String sequences..... 19
- 10.4 Restrictions on attributes..... 20
- 10.5 Restrictions on interleave..... 21
- 11 Conformance..... 21
- Annex A (normative) RELAX NG schema for RELAX NG..... 22
- Annex B (informative) Examples..... 28
- B.1 Data model..... 28
- B.2 Full syntax example..... 29
- B.3 Simple syntax example..... 30
- B.4 Validation example..... 30
- Bibliography..... 33

## Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

ISO/IEC 19757-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 34, Document Description and Processing Languages.

- *Part 1: Overview*
- *Part 2: Regular-grammar-based validation — RELAX NG*
- *Part 3: Rule-based validation — Schematron*
- *Part 4: Selection of validation candidates*
- *Part 5: Datatypes*
- *Part 6: Path-based integrity constraints*
- *Part 7: Character repertoire validation*
- *Part 8: Declarative document architecture*
- *Part 9: Datatype- and namespace-aware DTDs*
- *Part 10: Validation management*

## Introduction

The structure of is as follows. Clause 5 describes the data model, which is the abstraction of an XML document used throughout the rest of the document. Clause 6 describes the syntax of a RELAX NG schema. Clause 7 describes a sequence of transformations that are applied to simplify a RELAX NG schema, and also specifies additional requirements on a RELAX NG schema. Clause 8 describes the syntax that results from applying the transformations; this simple syntax is a subset of the full syntax. Clause 9 describes the semantics of a correct RELAX NG schema that uses the simple syntax; the semantics specify when an element is valid with respect to a RELAX NG schema. Clause 10 describes requirements that apply to a RELAX NG schema after it has been transformed into simple form. Finally, Clause 11 describes conformance requirements for RELAX NG validators.

is based on the RELAX NG Specification<sup>[1]</sup>. A tutorial for RELAX NG is available separately (see the RELAX NG Tutorial<sup>[2]</sup>).

# Document Schema Definition Languages (DSDL) — Part 2: Regular-grammar-based validation — RELAX NG

## 1 Scope

specifies RELAX NG, a schema language for XML. A RELAX NG schema specifies a pattern for the structure and content of an XML document. The pattern is specified by using a regular tree grammar. establishes requirements for RELAX NG schemas and specifies when an XML document matches the pattern specified by a RELAX NG schema.

## 2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 19757. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 19757 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE Each of the following documents has a unique identifier that is used to cite the document in the text. The unique identifier consists of the part of the reference up to the first comma.

W3C XML, *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, 6 October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>

W3C XML-Names, *Namespaces in XML*, W3C Recommendation, 14 January 1999, <http://www.w3.org/TR/1999/REC-xml-names-19990114/>

W3C XLink, *XML Linking Language (XLink) Version 1.0*, W3C Recommendation, 27 June 2001, <http://www.w3.org/TR/2001/REC-xlink-20010627/>

W3C XML-Infoset, *XML Information Set*, W3C Recommendation, 24 October 2001, <http://www.w3.org/TR/2001/REC-xml-infoset-20011024/>

IETF RFC 2045, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, Internet Standards Track Specification, November 1996, <http://www.ietf.org/rfc/rfc2045.txt>

IETF RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, Internet Standards Track Specification, November 1996, <http://www.ietf.org/rfc/rfc2046.txt>

IETF RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*, Internet Standards Track Specification, August 1998, <http://www.ietf.org/rfc/rfc2396.txt>

IETF RFC 2732, *Format for Literal IPv6 Addresses in URL's*, Internet Standards Track Specification, December 1999, <http://www.ietf.org/rfc/rfc2732.txt>

IETF RFC 3023, *XML Media Types*, Internet Standards Track Specification, August 1998, <http://www.ietf.org/rfc/rfc3023.txt>

### 3 Terms and definitions

#### 4 Notation

##### 4.1 EBNF

uses EBNF notation to describe the full syntax and the simple syntax of RELAX NG. A description of a grammar in EBNF consists of one or more production rules. Each production rule consists of the name of a non-terminal, followed by ::=, followed by a list of alternatives separated by |. Within an alternative, italic type is used to reference a non-terminal, concatenation indicates sequencing, [] indicates optionality, + indicates repetition one or more times and \* indicates repetition zero or more times; other characters in normal type stand for themselves.

##### 4.2 Inference rules

###### 4.2.1 Variables

The symbol used for a variable indicates the variable's range as follows:

- ranges over names
- ranges over name classes
- ranges over local names; a local name is a string that matches the NCName production of W3C XML-Names, that is, a name with no colons
- ranges over URIs
- ranges over contexts (as defined in Clause 5)
- ranges over sets of attributes; a set with a single member is considered the same as that member
- ranges over sequences of elements and strings; a sequence with a single member is considered the same as that member; the sequences ranged over by *m* may contain consecutive strings and may contain strings that are empty  
NOTE There are sequences ranged over by *m* that cannot occur as the children of an element.
- ranges over patterns (elements matching the pattern production)
- ranges over strings
- ranges over the empty sequence and strings that consist entirely of whitespace
- ranges over sequences of parameters
- ranges over elements
- ranges over content-types

###### 4.2.2 Propositions

The following notation is used for propositions:

- means that name *n* is a member of name class *nc*
- means that with respect to context *cx*, the attributes *a* and the sequence of elements and strings *m* matches the pattern *p*

- means that there is no name that is the name of both an attribute in  $a_1$  and of an attribute in  $a_2$
- means that  $m_1$  is an interleaving of  $m_2$  and  $m_3$
- means that with respect to context  $cx$ , the attributes  $a$  and the sequence of elements and strings  $m$  weakly matches the pattern  $p$
- means that the mixed sequence  $m$  can occur as the children of an element: it does not contain any member that is an empty string, nor does it contain two consecutive members that are both strings
- means that the grammar contains `<define name="ln"> <element> nc p </element> </define>`
- means that in the datatype library identified by URI  $u$ , the string  $s$  interpreted with context  $cx$  is a legal value of datatype  $ln$  with parameters  $params$
- means that in the datatype library identified by URI  $u$ , string  $s_1$  interpreted with context  $cx_1$  represents the same value of the datatype  $ln$  as the string  $s_2$  interpreted in the context of  $cx_2$
- means that  $s_1$  and  $s_2$  are identical
- means that the element  $e$  is valid with respect to the grammar
- means that the grammar contains `<start> p </start>`
- means that the content-types  $ct_1$  and  $ct_2$  are groupable
- means that pattern  $p$  has content-type  $ct$
- means that the schema is incorrect

### 4.2.3 Expressions

The following notation is used for expressions in propositions:

- returns a name with URI  $u$  and local name  $ln$
- returns the concatenation of the sequences  $m_1$  and  $m_2$
- returns the union of  $a_1$  and  $a_2$
- returns an empty sequence
- returns an empty set
- returns an empty string
- returns an attribute with name  $n$  and value  $s$
- returns an element with name  $n$ , context  $cx$ , attributes  $a$  and mixed sequence  $m$  as children
- returns the maximum of  $ct_1$  and  $ct_2$  where the content-types in increasing order are `empty( )`, `complex( )`, `simple( )`
- returns the string  $s$ , with leading and trailing whitespace characters removed, and with each other maximal sequence of whitespace characters replaced by a single space character

- returns a sequence of strings one for each whitespace delimited token of *s*; each string in the returned sequence will be non-empty and will not contain any whitespace
- returns a context which is the same as *cx* except that the default namespace is *u*; if *u* is the empty string, then there is no default namespace in the constructed context
- returns the empty content-type
- returns the complex content-type
- returns the simple content-type
- within the start-tag of a pattern refers to the context of the pattern element

### 5 Data model

RELAX NG deals with XML documents representing both schemas and instances through an abstract data model. XML documents representing schemas and instances shall be well-formed in conformance with W3C XML and shall conform to the constraints of W3C XML-Names.

An XML document is represented by an element. An element consists of

- a name
- a context
- a set of attributes
- an ordered sequence of zero or more children; each child is either an element or a non-empty string; the sequence never contains two consecutive strings

A name consists of

- a string representing the namespace URI; the empty string has special significance, representing the absence of any namespace
- a string representing the local name; this string matches the NCName production of W3C XML-Names

A context consists of

- a base URI
- a namespace map; this maps prefixes to namespace URIs, and also may specify a default namespace URI (as declared by the `xmlns` attribute)

An attribute consists of

- a name
- a string representing the value

A string consists of a sequence of zero or more characters, where a character is as defined in W3C XML.

The element for an XML document is constructed from the infoset (see W3C XML-Infoset) of the XML document as follows. The notation [x] refers to the value of the x property of an information item. An element is constructed from a document information item by constructing an element from the [document element]. An element is constructed from an element information item by constructing the name from the [namespace name] and [local name], the context from the [base URI] and [in-scope namespaces], the attributes from the [attributes], and the children from the [children]. The attributes of an element are constructed from the unordered set of attribute information items by constructing an attribute for each attribute information item. The children of an element are constructed from the list of child information items first by removing information items other than element information items and character information items, and then by constructing an element for each element information item in the list and a string for each maximal sequence of character information items. An attribute is constructed from an attribute information item by constructing the name from the [namespace name] and [local name], and the value from the [normalized value]. When constructing the name of an element or attribute from the [namespace name] and [local name], if the [namespace name] property is not present, then the name is constructed from an empty string and the [local name]. A string is constructed from a sequence of character information items by constructing a character from the [character code] of each character information item.

It is possible for there to be multiple distinct infosets for a single XML document. This is because XML parsers are not required to process all DTD declarations or expand all external parsed general entities. Amongst these multiple infosets, there is exactly one infoset for which [all declarations processed] is true and which does not contain any unexpanded entity reference information items. This is the infoset that is the basis for defining the RELAX NG data model.

## 6 Full syntax

The following grammar in EBNF notation summarizes the syntax of RELAX NG. Although the notation is based on the XML representation of an RELAX NG schema as a sequence of characters, the grammar operates at the data model level. For example, although the syntax uses `<text/>`, an instance or schema can use `<text></text>` instead, because they both represent the same element at the data model level. All elements shown in the grammar are qualified with the namespace URI:

```
http://relaxng.org/ns/structure/1.0
```

The symbols QName and NCName are defined in W3C XML-Names. The anyURI symbol indicates a string that, after escaping of disallowed values as described in Section 5.4 of W3C XLink, is a URI reference as defined in IETF RFC 2396 (as modified by IETF RFC 2732). The symbol string matches any string.

In addition to the attributes shown explicitly, any element can have an ns attribute and any element can have a datatypeLibrary attribute. The ns attribute can have any value. The value of the datatypeLibrary attribute shall match the anyURI symbol as described in the previous paragraph; in addition, it shall not use the relative form of URI reference and shall not have a fragment identifier; as an exception to this, the value may be the empty string.

Any element can also have foreign attributes in addition to the attributes shown in the grammar. A foreign attribute is an attribute with a name whose namespace URI is neither the empty string nor the RELAX NG namespace URI. Any element that cannot have string children (that is, any element other than value, param and name) may have foreign child elements in addition to the child elements shown in the grammar. A foreign element is an element with a name whose namespace URI is not the RELAX NG namespace URI. There are no constraints on the relative position of foreign child elements with respect to other child elements.

Any element can also have as children strings that consist entirely of whitespace characters, where a whitespace character is one of #x20, #x9, #xD or #xA. There are no constraints on the relative position of whitespace string children with respect to child elements.

Leading and trailing whitespace is allowed for value of each name, type and combine attribute and for the content of each name element.

```
pattern ::=
  <element name="QName"> pattern+ </element>
  | <element> nameClass pattern+ </element>
  | <attribute name="QName"> [pattern] </attribute>
```

```

| <attribute> nameClass [pattern] </attribute>
| <group> pattern+ </group>
| <interleave> pattern+ </interleave>
| <choice> pattern+ </choice>
| <optional> pattern+ </optional>
| <zeroOrMore> pattern+ </zeroOrMore>
| <oneOrMore> pattern+ </oneOrMore>
| <list> 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
  | <div> grammarContent* </div>
  | <include href="anyURI"> includeContent* </include>
includeContent ::=
  start
  | define
  | <div> includeContent* </div>
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>

```

## 7 Simplification

### 7.1 General

The full syntax given in the previous clause is transformed into a simpler syntax by applying the following transformation rules in order. The effect shall be as if each rule was applied to all elements in the schema before the next rule is applied. A transformation rule may also specify constraints that shall be satisfied by a correct schema. The transformation rules are applied at the data model level. Before the transformations are applied, the schema is parsed into an element in the data model.

### 7.2 Annotations

Foreign attributes and elements are removed.

**NOTE** It is safe to remove `xml:base` attributes at this stage because `xml:base` attributes are used in determining the [base URI] of an element information item, which is in turn used to construct the base URI of the context of an element. Thus, after a document has been parsed into an element in the data model, `xml:base` attributes can be discarded.

### 7.3 Whitespace

For each element other than `value` and `param`, each child that is a string containing only whitespace characters is removed.

Leading and trailing whitespace characters are removed from the value of each `name`, `type` and `combine` attribute and from the content of each `name` element.

### 7.4 datatypeLibrary attribute

The value of each `datatypeLibrary` attribute is transformed by escaping disallowed characters as specified in Section 5.4 of W3C XLink.

For any `data` or `value` element that does not have a `datatypeLibrary` attribute, a `datatypeLibrary` attribute is added. The value of the added `datatypeLibrary` attribute is the value of the `datatypeLibrary` attribute of the nearest ancestor element that has a `datatypeLibrary` attribute, or the empty string if there is no such ancestor. Then, any `datatypeLibrary` attribute that is on an element other than `data` or `value` is removed.

### 7.5 type attribute of value element

For any `value` element that does not have a `type` attribute, a `type` attribute is added with a value of `token` and the value of the `datatypeLibrary` attribute is changed to the empty string.

### 7.6 href attribute

The value of the `href` attribute on an `externalRef` or `include` element is first transformed by escaping disallowed characters as specified in Section 5.4 of W3C XLink. The URI reference is then resolved into an absolute form as described in Section 5.2 of IETF RFC 2396 using the base URI from the context of the element that bears the `href` attribute.

The value of the `href` attribute is used to construct an element (as specified in Clause 5). This shall be done as follows. The URI reference consists of the URI itself and an optional fragment identifier. The resource identified by the URI is retrieved. The result is a MIME entity (see IETF RFC 2045): a sequence of bytes labeled with a MIME media type (see IETF RFC 2046). The media type determines how an element is constructed from the MIME entity and optional fragment identifier. When the media type is `application/xml` or `text/xml`, the MIME entity shall be parsed as an XML document in accordance with the applicable RFC (at the term of writing IETF RFC 3023) and an element constructed from the result of the parse as specified in Clause 5. In particular, the `charset` parameter shall be handled as specified by the RFC. This specification does not define the handling of media types other than `application/xml` and `text/xml`. The `href` attribute shall not include a fragment identifier unless the registration of the media type of the resource identified by the attribute defines the interpretation of fragment identifiers for that media type.

**NOTE** IETF RFC 3023 does not define the interpretation of fragment identifiers for `application/xml` or `text/xml`.

### 7.7 externalRef element

An `externalRef` element is transformed as follows. An element is constructed using the URI reference that is the value of `href` attribute as specified in 7.6. This element shall match the syntax for pattern. The element is transformed by recursively applying the rules from this subclauses and from previous subclauses of this clause. This shall not result in a loop. In other words, the transformation of the referenced element shall not require the dereferencing of an `externalRef` element with an `href` attribute with the same value.

Any `ns` attribute on the `externalRef` element is transferred to the referenced element if the referenced element does not already have an `ns` attribute. The `externalRef` element is then replaced by the referenced element.

## 7.8 include element

An `include` element is transformed as follows. An element is constructed using the URI reference that is the value of `href` attribute as specified in 7.6. This element shall be a `grammar` element, matching the syntax for `grammar`.

This `grammar` element is transformed by recursively applying the rules from this subclause and from previous subclauses of this clause. This shall not result in a loop. In other words, the transformation of the `grammar` element shall not require the dereferencing of an `include` element with an `href` attribute with the same value.

Define the `children` of an element to be the children of the element together with the components of any `div` child elements. If the `include` element has a `start` component, then the `grammar` element shall have at least one `start` component; it is then transformed by removing all `start` components. If the `include` element has a `define` component, then the `grammar` element shall have at least one `define` component with the same name; it is then transformed by removing all such `define` components.

The `include` element is transformed into a `div` element. The attributes of the `div` element are the attributes of the `include` element other than the `href` attribute. The children of the `div` element are the `grammar` element (after the removal of the `start` and `define` components described by the preceding paragraph) followed by the children of the `include` element. The `grammar` element is then renamed to `div`.

## 7.9 name attribute of element and attribute elements

The `name` attribute on an element or attribute element is transformed into a `name` child element.

If an attribute element has a `name` attribute but no `ns` attribute, then an `ns=""` attribute is added to the `name` child element.

## 7.10 ns attribute

For any `name`, `nsName` or `value` element that does not have an `ns` attribute, an `ns` attribute is added. The value of the added `ns` attribute is the value of the `ns` attribute of the nearest ancestor element that has an `ns` attribute, or the empty string if there is no such ancestor. Then, any `ns` attribute that is on an element other than `name`, `nsName` or `value` is removed.

NOTE The value of the `ns` attribute is transformed either by escaping disallowed characters, or in any other way, because the value of the `ns` attribute is compared against namespace URIs in the instance, which are not subject to any transformation.

NOTE Since `include` and `externalRef` elements are resolved after `datatypeLibrary` attributes are added but before `ns` attributes are added, `ns` attributes are inherited into external schemas but `datatypeLibrary` attributes are not.

## 7.11 QName

For any `name` element containing a prefix, the prefix is removed and an `ns` attribute is added replacing any existing `ns` attribute. The value of the added `ns` attribute is the value to which the namespace map of the context of the `name` element maps the prefix. The context shall have a mapping for the prefix.

## 7.12 div element

Each `div` element is replaced by its children.

## 7.13 Number of child elements

A `define`, `oneOrMore`, `zeroOrMore`, `optional`, `list` or `mixed` element is transformed so that it has exactly one child element. If it has more than one child element, then its child elements are wrapped in a `group` element.

An `element` element is transformed so that it has exactly two child elements, the first being a name class and the second being a pattern. If it has more than two child elements, then the child elements other than the first are wrapped in a `group` element.

A `except` element is transformed so that it has exactly one child element. If it has more than one child element, then its child elements are wrapped in a `choice` element.

If an `attribute` element has only one child element (a `name class`), then a `text` element is added.

A `choice`, `group` or `interleave` element is transformed so that it has exactly two child elements. If it has one child element, then it is replaced by its child element. If it has more than two child elements, then the first two child elements are combined into a new element with the same name as the parent element and with the first two child elements as its children. For example,

```
<choice> p1 p2 p3 </choice>
```

is transformed to

```
<choice> <choice> p1 p2 </choice> p3 </choice>
```

This reduces the number of child elements by one. The transformation is applied repeatedly until there are exactly two child elements.

#### 7.14 mixed element

A `mixed` element is transformed into an interleaving with a `text` element:

```
<mixed> p </mixed>
```

is transformed into

```
<interleave> p <text/> </interleave>
```

#### 7.15 optional element

An `optional` element is transformed into a `choice` element with two children, one child being the child of the `optional` element and the other child being an `empty` element:

```
<optional> p </optional>
```

is transformed into

```
<choice> p <empty/> </choice>
```

#### 7.16 zeroOrMore element

A `zeroOrMore` element is transformed into a `choice` element with two children, one child being a `oneOrMore` element whose only child is the child of the `zeroOrMore` element and the other child being an `empty` element:

```
<zeroOrMore> p </zeroOrMore>
```

is transformed into

```
<choice> <oneOrMore> p </oneOrMore> <empty/> </choice>
```

#### 7.17 Constraints

In this rule, no transformation is performed, but various constraints are checked.

**NOTE** The constraints in this subclause, unlike the constraints specified in Clause 10, can be checked without resolving any `ref` elements, and are accordingly applied even to patterns that will disappear during later stages of simplification because they are not reachable (see 7.20) or because of `notAllowed` (see 7.21).

An `except` element that is a child of an `anyName` element shall not have any `anyName` descendant elements. An `except` element that is a child of an `nsName` element shall not have any `nsName` or `anyName` descendant elements.

A `name` element that occurs as the first child of an `attribute` element or as the descendant of the first child of an `attribute` element and that has an `ns` attribute with value equal to the empty string shall not have content equal to `xmlns`.

A `name` or `nsName` element that occurs as the first child of an `attribute` element or as the descendant of the first child of an `attribute` element shall not have an `ns` attribute with value `http://www.w3.org/2000/xmlns`.

NOTE The W3C XML-Infoset defines the namespace URI of namespace declaration attributes to be `http://www.w3.org/2000/xmlns`.

A `data` or `value` element shall be correct in its use of datatypes. Specifically, the `type` attribute shall identify a datatype within the datatype library identified by the value of the `datatypeLibrary` attribute. For a `data` element, the parameter list shall be one that is allowed by the datatype (see 9.3.8).

### 7.18 combine attribute

For each `grammar` element, all `define` elements with the same name are combined together. For any name, there shall not be more than one `define` element with that name that does not have a `combine` attribute. For any name, if there is a `define` element with that name that has a `combine` attribute with the value `choice`, then there shall not also be a `define` element with that name that has a `combine` attribute with the value `interleave`. Thus, for any name, if there is more than one `define` element with that name, then there is a unique value for the `combine` attribute for that name. After determining this unique value, the `combine` attributes are removed. A pair of definitions

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

is combined into

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

where `c` is the value of the `combine` attribute. Pairs of definitions are combined until there is exactly one `define` element for each name.

Similarly, for each `grammar` element all `start` elements are combined together. There shall not be more than one `start` element that does not have a `combine` attribute. If there is a `start` element that has a `combine` attribute with the value `choice`, there shall not also be a `start` element that has a `combine` attribute with the value `interleave`.

### 7.19 grammar element

In this rule, the schema is transformed so that its top-level element is `grammar` and so that it has no other `grammar` elements.

Define the for an element to be the nearest ancestor `grammar` element. A `ref` element a `define` element if the value of their `name` attributes is the same and their in-scope grammars are the same. A `parentRef` element a `define` element if the value of their `name` attributes is the same and the in-scope grammar of the in-scope grammar of the

`parentRef` element is the same as the in-scope grammar of the `define` element. Every `ref` or `parentRef` element shall refer to a `define` element. A grammar shall have a `start` child element.

First, transform the top-level pattern  $p$  into `<grammar><start>p</start></grammar>`. Next, rename `define` elements so that no two `define` elements anywhere in the schema have the same name. To rename a `define` element, change the value of its `name` attribute and change the value of the `name` attribute of all `ref` and `parentRef` elements that refer to that `define` element. Next, move all `define` elements to be children of the top-level `grammar` element, replace each nested `grammar` element by the child of its `start` element and rename each `parentRef` element to `ref`.

## 7.20 `define` and `ref` elements

In this rule, the grammar is transformed so that every `element` element is the child of a `define` element, and the child of every `define` element is an `element` element.

First, remove any `define` element that is not `.`. A `define` element is reachable if there is reachable `ref` element referring to it. A `ref` element is reachable if it is the descendant of the `start` element or of a reachable `define` element. Now, for each `element` element that is not the child of a `define` element, add a `define` element to the `grammar` element, and replace the `element` element by a `ref` element referring to the added `define` element. The value of the `name` attribute of the added `define` element shall be different from value of the `name` attribute of all other `define` elements. The child of the added `define` element is the `element` element.

Define a `ref` element to be `if` it refers to a `define` element whose child is not an `element` element. For each `ref` element that is expandable and is a descendant of a `start` element or an `element` element, expand it by replacing the `ref` element by the child of the `define` element to which it refers and then recursively expanding any expandable `ref` elements in this replacement. This shall not result in a loop. In other words expanding the replacement of a `ref` element having a `name` with value  $n$  shall not require the expansion of `ref` element also having a `name` with value  $n$ . Finally, remove any `define` element whose child is not an `element` element.

## 7.21 `notAllowed` element

In this rule, the grammar is transformed so that a `notAllowed` element occurs only as the child of a `start` or `element` element. An `attribute`, `list`, `group`, `interleave`, or `oneOrMore` element that has a `notAllowed` child element is transformed into a `notAllowed` element. A `choice` element that has two `notAllowed` child elements is transformed into a `notAllowed` element. A `choice` element that has one `notAllowed` child element is transformed into its other child element. An `except` element that has a `notAllowed` child element is removed. The preceding transformations are applied repeatedly until none of them is applicable any more. Any `define` element that is no longer reachable is removed.

## 7.22 `empty` element

In this rule, the grammar is transformed so that an `empty` element does not occur as a child of a `group`, `interleave`, or `oneOrMore` element or as the second child of a `choice` element. A `group`, `interleave` or `choice` element that has two `empty` child elements is transformed into an `empty` element. A `group` or `interleave` element that has one `empty` child element is transformed into its other child element. A `choice` element whose second child element is an `empty` element is transformed by interchanging its two child elements. A `oneOrMore` element that has an `empty` child element is transformed into an `empty` element. The preceding transformations are applied repeatedly until none of them is applicable any more.

## 8 Simple syntax

After applying all the rules in Clause 7, the schema will match the grammar described by the following EBNF notation:

```
grammar ::=
  <grammar> <start> top </start> define* </grammar>
define ::=
  <define name="NCName"> <element> nameClass top </element> </define>
top ::=
  <notAllowed/>
```

```

    | pattern
pattern ::=
    <empty/>
    | nonEmptyPattern
nonEmptyPattern ::=
    <text/>
    | <data type="NCName" datatypeLibrary="anyURI"> param* [exceptPattern] </data>
    | <value datatypeLibrary="anyURI" type="NCName" ns="string"> string </value>
    | <list> 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>

```

With this grammar, no elements or attributes are allowed other than those explicitly shown.

NOTE The simple syntax is purely an internal artifact of . The simple syntax is not an alternative interchange syntax for RELAX NG. A correct RELAX NG schema is always required to be in the full syntax and is never required to be in the simple syntax.

## 9 Semantics

### 9.1 Inference rules

This clause defines the semantics of a correct RELAX NG schema that has been transformed into the simple syntax. The semantics of a RELAX NG schema consist of a specification of what XML documents are valid with respect to that schema. The semantics are described formally. The formalism uses axioms and inference rules. Axioms are propositions that are provable unconditionally. An inference rule consists of one or more antecedents and exactly one consequent. An antecedent is either positive or negative. If all the positive antecedents of an inference rule are provable and none of the negative antecedents are provable, then the consequent of the inference rule is provable. An XML document is valid with respect to a RELAX NG schema if and only if the proposition that the element representing it in the data model is valid is provable in the formalism specified in this clause.

NOTE This kind of formalism is similar to a proof system. However, a traditional proof system only has positive antecedents.

The notation for inference rules separates the antecedents from the consequent by a horizontal line: the antecedents are above the line; the consequent is below the line. If an antecedent is of the form  $\text{not}(p)$ , then it is a negative antecedent; otherwise, it is a positive antecedent. Both axioms and inferences rules may use variables. A variable has a name and optionally a subscript. The name of a variable is italicized. Each variable has a range that is determined by its name. If an axiom or inference rule contains multiple variables with the same range, then subscripts are used to distinguish them. Axioms and inference rules are implicitly universally quantified over the variables they contain. We explain this further below.

The possibility that an inference rule or axiom may contain more than one occurrence of a particular variable requires that an identity relation be defined on each kind of object over which a variable can range. The identity relation for all kinds of object is value-based. Two objects of a particular kind are identical if the constituents of the objects are

identical. For example, two attributes are considered the same if they have the same name and the same value. Two characters are identical if their Unicode character codes are the same.

## 9.2 Name classes

The main semantic concept for name classes is that of a name belonging to a name class. A name class is an element that matches the production nameClass. A name is as defined in Clause 5: it consists of a namespace URI and a local name.

The first axiom is called (anyName 1):

(anyName 1)  $n \text{ in } \langle \text{anyName} \rangle$

NOTE The (anyName 1) axiom says that for any name  $n$ ,  $n$  belongs to the name class  $\langle \text{anyName} \rangle$ , in other words  $\langle \text{anyName} \rangle$  matches any name. Note the effect of the implicit universal quantification over the variables in the axiom: this is what makes the axiom apply for any name  $n$ .

The first inference rule is almost as simple:

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

NOTE The (anyName 2) inference rule says that for any name  $n$  and for any name class  $nc$ , if  $n$  does not belong to  $nc$ , then  $n$  belongs to  $\langle \text{anyName} \rangle \langle \text{except} \rangle nc \langle / \text{except} \rangle \langle / \text{anyName} \rangle$ . In other words,  $\langle \text{anyName} \rangle \langle \text{except} \rangle nc \langle / \text{except} \rangle \langle / \text{anyName} \rangle$  matches any name that does not match  $nc$ .

The remaining axioms and inference rules for name classes are as follows:

(nsName 1)  $\text{name}(u, ln) \text{ in } \langle \text{nsName } ns="u" \rangle$

(nsName 2) 
$$\frac{\text{not}(\text{name}(u, ln) \text{ in } nc)}{\text{name}(u, ln) \text{ in } \langle \text{nsName } ns="u" \rangle \langle \text{except} \rangle nc \langle / \text{except} \rangle \langle / \text{nsName} \rangle}$$

(name)  $\text{name}(u, ln) \text{ in } \langle \text{name } ns="u" \rangle ln \langle / \text{name} \rangle$

(name choice 1) 
$$\frac{n \text{ in } nc_1}{n \text{ in } \langle \text{choice} \rangle nc_1 nc_2 \langle / \text{choice} \rangle}$$

(name choice 2) 
$$\frac{n \text{ in } nc_2}{n \text{ in } \langle \text{choice} \rangle nc_1 nc_2 \langle / \text{choice} \rangle}$$

## 9.3 Patterns

### 9.3.1 choice pattern

The semantics of the choice pattern are as follows:

(choice 1) 
$$\frac{cx \text{ |- } a; m \sim p_1}{cx \text{ |- } a; m \sim \langle \text{choice} \rangle p_1 p_2 \langle / \text{choice} \rangle}$$

(choice 2) 
$$\frac{cx \text{ |- } a; m \sim p_2}{cx \text{ |- } a; m \sim \langle \text{choice} \rangle p_1 p_2 \langle / \text{choice} \rangle}$$

### 9.3.2 group pattern

The semantics of the group pattern are as follows:

$$\text{(group)} \quad \frac{cx \mid - a_1; m_1 = \sim p_1 \quad cx \mid - a_2; m_2 = \sim p_2}{cx \mid - a_1 + a_2; m_1, m_2 = \sim \langle \text{group} \rangle p_1 p_2 \langle / \text{group} \rangle}$$

NOTE The restriction in 10.4 ensures that the set of attributes constructed in the consequent will not have multiple attributes with the same name.

**9.3.3 empty pattern**

The semantics of the `empty` pattern are as follows:

$$\text{(empty)} \quad cx \mid - \{ \}; ( ) = \sim \langle \text{empty} \rangle / >$$

**9.3.4 text pattern**

The semantics of the `text` pattern are as follows:

$$\text{(text 1)} \quad cx \mid - \{ \}; ( ) = \sim \langle \text{text} \rangle / >$$

$$\text{(text 2)} \quad \frac{cx \mid - \{ \}; m = \sim \langle \text{text} \rangle / >}{cx \mid - \{ \}; m, s = \sim \langle \text{text} \rangle / >}$$

The effect of the above rule is that a `text` element matches zero or more strings.

**9.3.5 oneOrMore pattern**

The semantics of the `oneOrMore` pattern are as follows:

$$\text{(oneOrMore 1)} \quad \frac{cx \mid - a; m = \sim p}{cx \mid - a; m = \sim \langle \text{oneOrMore} \rangle p \langle / \text{oneOrMore} \rangle}$$

$$\text{(oneOrMore 2)} \quad \frac{cx \mid - a_1; m_1 = \sim p \quad cx \mid - a_2; m_2 = \sim \text{disjoint}(a_1, a_2) \langle \text{oneOrMore} \rangle p \langle / \text{oneOrMore} \rangle}{cx \mid - a_1 + a_2; m_1, m_2 = \sim \langle \text{oneOrMore} \rangle p \langle / \text{oneOrMore} \rangle}$$

**9.3.6 interleave pattern**

The semantics of interleaving are defined by the following rules.

$$\text{(interleaves 1)} \quad ( ) \text{ interleaves } ( ); ( )$$

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

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

For example, the interleavings of `<a/><a/>` and `<b/>` are `<a/><a/><b/>`, `<a/><b/><a/>`, and `<b/><a/><a/>`.

The semantics of the `interleave` pattern are as follows:

$$\text{(interleave)} \quad \frac{cx \mid - a_1; m_1 = \sim p_1 \quad cx \mid - a_2; m_2 = \sim p_2 \quad m_3 \text{ interleaves } m_1; m_2}{cx \mid - a_1 + a_2; m_3 = \sim \langle \text{interleave} \rangle p_1 p_2 \langle / \text{interleave} \rangle}$$

NOTE The restriction in 10.4 ensures that the set of attributes constructed in the consequent will not have multiple attributes with the same name.

**9.3.7 element and attribute pattern**

The value of an attribute is always a single string, which may be empty. Thus, the empty sequence is not a possible attribute value. On the other hand, the children of an element can be an empty sequence and cannot consist of an empty string. In order to ensure that validation handles attributes and elements consistently, a variant of matching called is used. Weak matching is used when matching the pattern for the value of an attribute or for the attributes and children of an element.

The semantics of weak matching are as follows:

(weak match 1)	$\frac{cx \mid - a; m = \sim p}{cx \mid - a; m = \sim_{\text{weak}} p}$
(weak match 2)	$\frac{cx \mid - a; () = \sim p}{cx \mid - a; ws = \sim_{\text{weak}} p}$
(weak match 3)	$\frac{cx \mid - a; " " = \sim p}{cx \mid - a; () = \sim_{\text{weak}} p}$

The semantics of the attribute pattern are as follows:

(attribute)	$\frac{cx \mid - \{ \}; s = \sim_{\text{weak}} p \qquad n \text{ in } nc}{cx \mid - \text{attribute}( n, s ); () = \sim \langle \text{attribute} \rangle nc p \langle / \text{attribute} \rangle}$
-------------	--

The semantics of the element pattern are as follows:

(element)	$\frac{cx_1 \mid - a; m = \sim_{\text{weak}} p \qquad n \text{ in } nc \quad \text{okAsChildren}(m) \quad \text{deref}(ln) = \langle \text{element} \rangle nc p \langle / \text{element} \rangle}{cx_2 \mid - \{ \}; ws_1, \text{element}( n, cx_1, a, m ), ws_2 = \sim \langle \text{ref name} = "ln" / \rangle}$
-----------	---

**9.3.8 data and value pattern**

RELAX NG relies on datatype libraries to perform datatyping. A datatype library is identified by a URI. A datatype within a datatype library is identified by an NCName. A datatype library provides two services.

- It can determine whether a string is a legal representation of a datatype. This service accepts a list of zero or more parameters. For example, a string datatype might have a parameter specifying the length of a string. The datatype library determines what parameters are applicable for each datatype.
- It can determine whether two strings represent the same value of a datatype. This service does not have any parameters.

Both services may make use of the context of a string. For example, a datatype representing a QName would use the namespace map.

The datatypeEqual function shall be reflexive, transitive and symmetric, that is, the following inference rules shall hold:

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

$$\begin{array}{l}
 \text{(datatypeEqual transitive)} \\
 \text{(datatypeEqual symmetric)}
 \end{array}
 \frac{\text{datatypeEqual}(u, ln, s_1, cx_1, s_2, cx_2) \quad \text{datatypeEqual}(u, ln, s_2, cx_2, s_3, cx_3)}{\text{datatypeEqual}(u, ln, s_1, cx_1, s_3, cx_3)}$$

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

The semantics of the data and value patterns are as follows:

$$\begin{array}{l}
 \text{(value)} \\
 \text{(data 1)} \\
 \text{(data 2)}
 \end{array}
 \frac{\text{datatypeEqual}(u_1, ln, s_1, cx_1, s_2, \text{context}(u_2, cx_2))}{cx_1 \mid \{ \}; s_1 = \sim \langle \text{value datatypeLibrary} = "u_1" \text{ type} = "ln" \text{ ns} = "u_2" [cx_2] \rangle s_2 \langle / \text{value} \rangle}$$

$$\frac{\text{datatypeAllows}(u, ln, params, s, cx)}{cx \mid \{ \}; s = \sim \langle \text{data datatypeLibrary} = "u" \text{ type} = "ln" \rangle params \langle / \text{data} \rangle}$$

$$\frac{\text{datatypeAllows}(u, ln, params, s, cx) \quad \text{not}(cx \mid a; s = \sim p)}{cx \mid \{ \}; s = \sim \langle \text{data datatypeLibrary} = "u" \text{ type} = "ln" \rangle params \langle \text{except} \rangle p \langle / \text{except} \rangle \langle / \text{data} \rangle}$$

### 9.3.9 Built-in datatype library

The empty URI identifies a special built-in datatype library. This provides two datatypes, string and token. No parameters are allowed for either of these datatypes.

The semantics of the two built-in datatypes are as follows:

$$\begin{array}{l}
 \text{(string allows)} \\
 \text{(string equal)} \\
 \text{(token allows)} \\
 \text{(token equal)}
 \end{array}
 \frac{\text{datatypeAllows}("", "string", (), s, cx)}{\text{datatypeEqual}("", "string", s, cx_1, s, cx_2)}$$

$$\frac{\text{datatypeAllows}("", "token", (), s, cx)}{\text{normalizeWhiteSpace}(s_1) = \text{normalizeWhiteSpace}(s_2)}$$

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

### 9.3.10 list pattern

The semantics of the list pattern are as follows:

$$\text{(list)} \quad \frac{cx \mid \{ \}; \text{split}(s) = \sim p}{cx \mid \{ \}; s = \sim \langle \text{list} \rangle p \langle / \text{list} \rangle}$$

NOTE It is crucial in the above inference rule that the sequence that is matched against a pattern can contain consecutive strings.

## 9.4 Validity

An element is valid with respect to a schema if the element together with an empty set of attributes it matches the start pattern of the schema's grammar.

$$\text{(valid)} \quad \frac{\text{start}() = p \quad cx \mid \{ \}; e = \sim p}{\text{valid}(e)}$$

## 10 Restrictions

### 10.1 General

The following constraints are all checked after the grammar has been transformed to the simple form described in Clause 8.

NOTE The purpose of these restrictions is to catch user errors and to facilitate implementation.

### 10.2 Prohibited paths

#### 10.2.1 General

This clause describes restrictions on where elements are allowed in the schema based on the names of the ancestor elements. The concept of a path is used to describe these restrictions. A path is a sequence of NCNames separated by / or //.

- An element matches a path  $x$ , where  $x$  is an NCName, if and only if the local name of the element is  $x$
- An element matches a path  $x/p$ , where  $x$  is an NCName and  $p$  is a path, if and only if the local name of the element is  $x$  and the element has a child that matches  $p$
- An element matches a path  $x//p$ , where  $x$  is an NCName and  $p$  is a path, if and only if the local name of the element is  $x$  and the element has a descendant that matches  $p$

For example, the element

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

matches the paths `foo`, `foo/bar`, `foo//bar`, `foo//baz`, `foo/bar/baz`, `foo/bar//baz` and `foo//bar/baz`, but not `foo/baz` or `foobar`.

A correct RELAX NG schema shall be such that, after transformation to the simple form, it does not contain any element that matches a prohibited path.

#### 10.2.2 attribute pattern

The following paths are prohibited:

- `attribute//ref`
- `attribute//attribute`

#### 10.2.3 oneOrMore pattern

The following paths are prohibited:

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

#### 10.2.4 list pattern

The following paths are prohibited:

- list//list
- list//ref
- list//attribute
- list//text
- list//interleave

#### 10.2.5 except element in data pattern

The following paths are prohibited:

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

NOTE This implies that an `except` element with a `data` parent can contain only data, value and choice elements.

#### 10.2.6 start element

The following paths are prohibited:

- start//attribute
- start//data
- start//value
- start//text
- start//list
- start//group
- start//interleave
- start//oneOrMore
- start//empty

### 10.3 String sequences

RELAX NG does not allow a pattern such as:

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

Nor does it allow a pattern such as:

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

**NOTE** For clarity, the above two examples have not been transformed to the simple syntax. This does not affect the requirement that the constraint in this sub-clause, as with all other constraints in this clause, is checked after the grammar has been transformed to the simple syntax.

More generally, if the pattern for the content of an element or attribute contains

- a pattern that can match a child (that is, an `element`, `data`, `value`, `list` or `text` pattern), and
- a pattern that matches a single string (that is, a `data`, `value` or `list` pattern),

then the two patterns shall be alternatives to each other.

This rule does not apply to patterns occurring within a `list` pattern.

To formalize this, the concept of a content-type is used. A pattern that is allowable as the content of an element has one of three content-types: empty, complex and simple.

The empty content-type is groupable with anything. In addition, the complex content-type is groupable with the complex content-type. The following rules formalize this.

(group empty 1)	<code>groupable(empty( ), ct)</code>
(group empty 2)	<code>groupable(ct, empty( ))</code>
(group complex)	<code>groupable(complex( ), complex( ))</code>

Some patterns have a content-type. The following rules define when a pattern has a content-type and, if so, what it is.

(value)	<code>&lt;value datatypeLibrary="u<sub>1</sub>" type="ln" ns="u<sub>2</sub>"&gt; s &lt;/value&gt; :<sub>c</sub> simple( )</code>
(data 1)	<code>&lt;data datatypeLibrary="u" type="ln"&gt; params &lt;/data&gt; :<sub>c</sub> simple( )</code>

	$p :_c ct$
(data 2)	$\frac{\langle data \ datatypeLibrary="u" \ type="ln" \rangle \ params \ \langle except \rangle \ p \ \langle /except \rangle \ \langle /data \rangle :_c \ simple( )}{}$
(list)	$\langle list \rangle \ p \ \langle /list \rangle :_c \ simple( )$
(text)	$\langle text \ / \rangle :_c \ complex( )$
(ref)	$\langle ref \ name="ln" \ / \rangle :_c \ complex( )$
(empty)	$\langle empty \ / \rangle :_c \ empty( )$
(attribute)	$\frac{p :_c ct}{\langle attribute \rangle \ nc \ p \ \langle /attribute \rangle :_c \ empty( )}$
(group)	$\frac{p_1 :_c ct_1 \quad p_2 :_c ct_2 \quad groupable(ct_1, ct_2)}{\langle group \rangle \ p_1 \ p_2 \ \langle /group \rangle :_c \ max( ct_1, ct_2 )}$
(interleave)	$\frac{p_1 :_c ct_1 \quad p_2 :_c ct_2 \quad groupable(ct_1, ct_2)}{\langle interleave \rangle \ p_1 \ p_2 \ \langle /interleave \rangle :_c \ max( ct_1, ct_2 )}$
(oneOrMore)	$\frac{p :_c ct \quad groupable(ct, ct)}{\langle oneOrMore \rangle \ p \ \langle /oneOrMore \rangle :_c \ ct}$
(choice)	$\frac{p_1 :_c ct_1 \quad p_2 :_c ct_2}{\langle choice \rangle \ p_1 \ p_2 \ \langle /choice \rangle :_c \ max( ct_1, ct_2 )}$

NOTE The antecedent in the (data 2) rule above is in fact redundant because of the prohibited paths in 10.2.5.

Now the restriction can be described. All patterns occurring as the content of an element pattern shall have a content-type.

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

### 10.4 Restrictions on attributes

Duplicate attributes are not allowed. More precisely, for a pattern  $\langle group \rangle \ p1 \ p2 \ \langle /group \rangle$  or  $\langle interleave \rangle \ p1 \ p2 \ \langle /interleave \rangle$ , there shall not be a name that belongs to both the name class of an attribute pattern occurring in  $p1$  and the name class of an attribute pattern occurring in  $p2$ . A pattern  $p1$  is defined to a pattern  $p2$  if

- $p1$  is  $p2$ , or
- $p2$  is a choice, interleave, group or oneOrMore element and  $p1$  occurs in one or more children of  $p2$ .

Attributes using infinite name classes shall be repeated. More precisely, an attribute element that has an anyName or nsName descendant element shall have a oneOrMore ancestor element.

NOTE This restriction is necessary for closure under negation.

Attributes using infinite name classes shall have `text` as their value. More precisely, an `attribute` element that has an `anyName` or `nsName` descendant element shall have a `text` child element.

NOTE This restriction is necessary for closure under negation.

### 10.5 Restrictions on `interleave`

In order to facilitate implementation, an element of a particular name shall be allowed by at most one operand of an `interleave` pattern; similarly, a `text` pattern shall occur in at most one operand of an `interleave` pattern. More precisely, for a pattern `<interleave> p1 p2 </interleave>`,

- there shall not be a name that belongs to both the name class of an `element` pattern referenced by a `ref` pattern occurring in `p1` and the name class of an `element` pattern referenced by a `ref` pattern occurring in `p2`, and
- a `text` pattern shall not occur in both `p1` and `p2`.

10.4 defines when one pattern is considered to occur in another pattern.

## 11 Conformance

A conforming RELAX NG validator shall be able to determine for any XML document whether it is a correct RELAX NG schema. A conforming RELAX NG validator shall be able to determine for any XML document and for any correct RELAX NG schema whether the document is valid with respect to the schema.

However, the requirements in the preceding paragraph do not apply if the schema uses a datatype library that the validator does not support. A conforming RELAX NG validator is only required to support the built-in datatype library described in 9.3.9. A validator that claims conformance to RELAX NG should document which datatype libraries it supports. The requirements in the preceding paragraph also do not apply if the schema includes `externalRef` or `include` elements and the validator is unable to retrieve the resource identified by the URI or is unable to construct an element from the retrieved resource. A validator that claims conformance to RELAX NG should document its capabilities for handling URI references.

## Annex A (normative)

### RELAX NG schema for RELAX NG

A correct RELAX NG schema shall be valid with respect to the following schema.

```

<grammar datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
  ns="http://relaxng.org/ns/structure/1.0"
  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="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">
    <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="common-atts">
  <optional>
    <attribute name="ns"/>
  </optional>
  <optional>
    <attribute name="datatypeLibrary">
      <data type="anyURI"/>
    </attribute>
  </optional>
  <zeroOrMore>
    <attribute>
      <anyName>
        <except>
          <nsName/>
        </except>
      </anyName>
    </attribute>
  </zeroOrMore>
</define>

```

```

        <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>

```

## Annex B (informative)

### Examples

#### B.1 Data model

Suppose the document `http://www.example.com/doc.xml` is as follows:

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

The element representing this document has

- a name which has
  - the empty string as the namespace URI, representing the absence of any namespace
  - `foo` as the local name
- a context which has
  - `http://www.example.com/doc.xml` as the base URI
  - a namespace map which
    - maps the prefix `xml` to the namespace URI `http://www.w3.org/XML/1998/namespace` (the `xml` prefix is implicitly declared by every XML document)
    - specifies the empty string as the default namespace URI
- an empty set of attributes
- a sequence of children consisting of an element which has
  - a name which has
    - `http://www.example.com/n1` as the namespace URI
    - `bar1` as the local name
  - a context which has
    - `http://www.example.com/doc.xml` as the base URI
    - a namespace map which
      - maps the prefix `pre1` to the namespace URI `http://www.example.com/n1`
      - maps the prefix `xml` to the namespace URI `http://www.w3.org/XML/1998/namespace`

- specifies the empty string as the default namespace URI
- an empty set of attributes
- an empty sequence of children

followed by an element which has

- a name which has
  - `http://www.example.com/n2` as the namespace URI
  - `bar2` as the local name
- a context which has
  - `http://www.example.com/doc.xml` as the base URI
  - a namespace map which
    - maps the prefix `pre2` to the namespace URI `http://www.example.com/n2`
    - maps the prefix `xml` to the namespace URI `http://www.w3.org/XML/1998/namespace`
    - specifies the empty string as the default namespace URI
- an empty set of attributes
- an empty sequence of children

## B.2 Full syntax example

Here is an example of a schema in the full syntax for the document in B.1.

```
<?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>
```

### B.3 Simple syntax example

The schema in B.2 could be transformed into the simple syntax:

```
<?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>
```

NOTE Strictly speaking, the result of simplification is an element in the data model rather than an XML document. For convenience, an XML document is used to represent an element in the data model.

### B.4 Validation example

Let  $e_0$  be

$$\text{element}(\text{name}(" ", "foo"), cx_0, \{\}, m)$$

where  $m$  is

$$e_1, e_2$$

and  $e_1$  is

$$\text{element}(\text{name}("http://www.example.com/n1", "bar1"), cx_1, \{\}, ( ))$$

and  $e_2$  is

$$\text{element}(\text{name}("http://www.example.com/n2", "bar2"), cx_2, \{\}, ( ))$$

Assuming appropriate definitions of  $cx_0$ ,  $cx_1$  and  $cx_2$ , this represents the document in B.1.

We now show how  $e_0$  can be shown to be valid with respect to the schema in B.3. The schema is equivalent to the following propositions:

$\text{start}() = \langle \text{ref name}="foo"/ \rangle$

$\text{deref}("foo.element") = \langle \text{element} \rangle \langle \text{name ns}="" \rangle "foo" \langle / \text{name} \rangle \langle \text{group} \rangle \langle \text{ref name}="bar1"/ \rangle \langle \text{ref name}="bar2"/ \rangle \langle / \text{group} \rangle \langle / \text{element} \rangle$

$\text{deref}("bar1.element") = \langle \text{element} \rangle \langle \text{name ns}="http://www.example.com/n1" \rangle "bar1" \langle / \text{name} \rangle \langle \text{empty}/ \rangle \langle / \text{element} \rangle$

$\text{deref}("bar2.element") = \langle \text{element} \rangle \langle \text{name ns}="http://www.example.com/n2" \rangle "bar2" \langle / \text{name} \rangle \langle \text{empty}/ \rangle \langle / \text{element} \rangle$

Let name class  $nc_1$  be

$\langle \text{name ns}="http://www.example.com/n1" \rangle "bar1" \langle / \text{name} \rangle$

and let  $nc_2$  be

$\langle \text{name ns}="http://www.example.com/n2" \rangle "bar2" \langle / \text{name} \rangle$

Then, by the inference rule (name) in 9.2, we have

$\text{name}("http://www.example.com/n1", "bar1") \text{ in } nc_1$

and

$\text{name}("http://www.example.com/n2", "bar2") \text{ in } nc_2$

By the inference rule (empty) in 9.3.3, we have

$cx_1 \vdash \{ \}; () \approx \langle \text{empty}/ \rangle$

and

$cx_2 \vdash \{ \}; () \approx \langle \text{empty}/ \rangle$

Thus by the inference rule (element) in 9.3.7, we have

$cx_0 \vdash \{ \}; e_1 \approx \langle \text{ref name}="bar1"/ \rangle$

Note that we have chosen  $cx_0$ , since any context is allowed.

Likewise, we have

$cx_0 \vdash \{ \}; e_2 \approx \langle \text{ref name}="bar2"/ \rangle$

By the inference rule (group) in 9.3.1, we have

$cx_0 \vdash \{ \}; e_1, e_2 \approx \langle \text{group} \rangle \langle \text{ref name}="bar1"/ \rangle \langle \text{ref name}="bar2"/ \rangle \langle / \text{group} \rangle$

By the inference rule (element) in 9.3.7, we have

$cx_3 \vdash \{ \}; \text{element}(\text{name}("", "foo"), cx_0, \{ \}, m) \approx \langle \text{ref name}="foo"/ \rangle$

Here  $cx_3$  is an arbitrary context.

Thus we can apply the inference rule (valid) in 9.4 and obtain

$\text{valid}(e_0)$

## Bibliography

- [1] *RELAX NG Specification*, OASIS Committee Specification, 3 December 2001,  
<http://www.oasis-open.org/committees/relax-ng/spec-20011203.html>
- [2] *RELAX NG Tutorial*, OASIS Committee Specification, 3 December 2001,  
<http://www.oasis-open.org/committees/relax-ng/tutorial-20011203.html>

Summary of editorial comments: