Notes on
Modeling XML Element Content Models

Harrie Passier

June 16, 2011

Abstract

We describe a method for modeling element content of XML-elements. Often, these models are simple, but complex ones occur. Because modeling the complex ones is not trivial, designers of these models may have problems with drawing up correct and concise models. As far as we know, study books about XML do not present methods for this type of modeling. The method we propose is based on modeling regular expressions. Using this, it is possible to model element content of XML-elements more systematically.

KEYWORDS: XML, modeling, regular expression, didactics of informatics

1 Introduction

DTD’s and content models

The type of an XML document is described by a grammar, for example a document type definition (DTD) or an XML-Schema. In these notes, we focus on DTD’s. Roughly, a DTD not only lists the elements an XML document consists of but also lists the attributes as part of each element. An element declaration looks like <!ELEMENT element-name content-model>. An example of a DTD is:

```
<!ELEMENT books (book+)>
<!ELEMENT book (title, (author+), (chapter+))>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT chapter (#PCDATA)>
```

There are four types of content models:

1. Empty. The element does not contain child elements. For example: <!ELEMENT cast EMPTY>.

2. Any. The content of the element can consist of any sequence of character data or elements. These elements must be declared by corresponding element declarations in the DTD. For example: <!ELEMENT cast ANY>.
3. Mixed. The content model has the form of $(\#\text{PCDATA} \mid e_1 \mid e_2 \ldots)^*$, where each $e_i$ is an element name. This means the content may contain arbitrary character data, interspersed with any number of elements of the specified names. For example $<!\text{ELEMENT} \text{cast} \ (\#\text{PCDATA} \mid \text{actor})^*>$.

4. Element content. The element contains a specific number of child elements in a specific order. For example $<!\text{ELEMENT} \text{cast} \ (\text{title}, \text{actor}^+)>$.

In our example, the elements books and book have a content model of type element content. The content model of element books is book+, which means one or more (+) book elements. Other cardinalities are *: zero or more times, and ?: zero or one times. The content model of the element book consists of a sequence of one title element, followed by one or more author elements, followed by one or more chapter elements. A sequence of elements is denoted by a comma separated list. Another possibility is a choice, depicted by a vertical bar |. For example, car*|cycle+ means zero or more car elements or one or more cycle elements. In the example above, the elements title, author, and chapter are of type parsed character data, i.e. of type string. Parsed character data (#PCDATA) is the text found between the start and end tag of an XML element. A possible XML instance document of the example DTD is:

```
<books>
  <book>
    <title>XML: Theory and applications</title>
    <author>L. Wiegering</author>
    <author>H. Pootjes</author>
    <author>H. Passier</author>
    <chapter>Introduction</chapter>
    <chapter>Basics of XML</chapter>
  </book>
</books>
```

In these notes we focus on the fourth type of content model, element content, and how to model this type systematically.

**Difficulties with modeling element content** In the example above, the element content of the elements books and book is simple. But in some cases finding the right model, i.e. representing the order and cardinalities of child elements, can be difficult. In the following example, the element content of element rec is more complex.

```
<recs>
  <rec>
    <a/>b/<c/>
  </rec>
  <rec>
    <b/>c/<c/>
  </rec>
</recs>
```
Each rec element contains a particular sequence of elements a, b, c and/or d. All these elements are empty, i.e. do not contain other elements and/or character data. A corresponding content model of element rec could be $(b, a?, c)|(a, b, c, d?)$. Finding this concise model by hand in an efficient way is not a trivial task and requires some knowledge and training.

**Existing educational material** As far as we know, there is no text book about XML that presents a systematic way for modeling content models of DTD’s (and/or XML-Schema’s). In one book, it is observed that element content can be written using a variation of the regular expression notation [16]. But it is not explained how to reach a correct and concise model.

**XML-editors** Most XML-editors contain an algorithm for inferring content models as part of a function for generating DTD’s or XML-Schema’s (see for example [1]). Often, these generated models can be made more concise. Consider, for example, the following XML-document:

```xml
<recs>
  <rec>
    <a/><b/>c/>
  </rec>
  <rec>
    <a/><b/>c/>
  </rec>
  <rec/>
  <rec>
    <a/><b/>c/>
  </rec>
</recs>
```

Using XMLSpy, the generated content model of element rec is $(a, b, (a, b, (a, b)?)?)$. This model is precise, i.e. only zero, one, two or three sequences of a- and b-elements are allowed. On the other hand, the model $(a, b)^*$ is more concise and often usable. The price we have paid is a decrease in precision, i.e. all sequences of a- en b-elements are allowed.

Furthermore, DTD-generators can be erroneous. For example, the content model of element b in the following XML-document
generated by XML-Spy (Enterprise edition version 2007) is \((c|d), c\)?, which is of course incorrect, since one c-element as content (the second b-element) is not part of the generated model: empty, \((c, c)\) and \((d, c)\). A correct one is for example \((d?, c)\)?. These examples make clear that manual inference remains necessary.

**Literature**  Approximately half of the XML documents available do not refer to a schema [2, 15]. Furthermore, about two-thirds of XML Schema definitions gathered from schema repositories and from the web at large are not valid with respect to the W3C XML Schema specification and are prove to be erroneous, rendering them essentially useless for immediate application [4, 14]. A similar observation was made concerning DTD’s [8].

**Approach**  We propose a method for modeling element content of XML-elements by rewriting element content using rules for manipulating regular expressions in a systematic way. This will help us finding a correct, precise, as well as a concise content model in cases where element content has a complex structure. We introduce the method for DTD-elements, but the method will be useful for modeling XML-Schema complex types with complex content too.

**These notes**  In these notes, we describe the approach in an informal style. In section 2 we introduce the concept regular expression, including the language and the rules for rewriting those expressions. Section 3 shows the relation between XML element content and regular expressions. In section 4 we show how the rules can be applied to rewrite element content. Section 5 discusses what deterministic content models are, and how to handle non deterministic ones. In section 6 we explain briefly what a rewrite system is, define the goals we want to reach in rewriting element content, and present a strategy for applying the rules more systematically. Section 7 formalizes the approach described. Section 8 briefly describes the main idea behind the algorithms used in most XML-editors. Finally, section 9 concludes and summarizes the results of these notes.
2 Regular expressions

In this section we introduce the concept of regular expressions and the rules for rewriting such expressions in a certain form. Regular expressions are an important formalism used in most schema languages for describing element content as part of the content model of XML-elements [16].

2.1 Definition

Definition A regular expression (RE) over an alphabet \( \Sigma \) consisting of some set of atoms, that are typically characters or names, is based on the following rules [11]:

1. the empty set \( \emptyset \) is a regular expression;
2. the empty string \( \epsilon \) is a regular expression;
3. each atom in \( \Sigma \) is by itself a regular expression;
4. if \( R \) and \( S \) are regular expressions, then the following are also regular expressions:
   \( R? \), \( R* \), \( R+ \), \( RS \), \( R|S \), and \( (R) \)

where the unary operators \?, *\, + express cardinalities, i.e. zero or one time (?), zero or more times (*) and one or more times (+). The binary operator \|\ means a choice between two expressions. Concatenation of two expressions is denoted by juxtaposition. The operators ?, *, + bind stronger than concatenation, and concatenation binds stronger than the choice operator. For example, the expression \( ab*|c \), where the alphabet is \( a, b, c \), must be interpreted as \( (a(b*))|c \) and not as \( a(b*|c) \) or as \( (ab)^*|c \).

Remark The definitions \( R? \) and \( R^+ \) are actually redundant, because \( R? \) is equivalent with \( R|\epsilon \) and \( R^+ \) is equivalent with \( RR^* \).

2.2 The language of a regular expression

Not every regular expression describes a single string. Instead, a regular expression describes a language that is a set of strings. We now give the rules to associate every well formed regular expression with the language it represents.

Definition We define inductively the language \( L(R) \), represented by a regular expression \( R \), as follows [12]:

\[
\begin{align*}
L(\emptyset) & = \emptyset \\
L(\epsilon) & = \{\epsilon\} \\
L(a) & = \{a\} \\
L(ST) & = L(S)L(T) \\
L(S|T) & = L(S) \cup L(T) \\
L(T^*) & = (L(T))^* \\
L(T?) & = L(T|\epsilon) = L(T) \cup \{\epsilon\} \\
L(T^+) & = L(TT^*) = L(T)L(T^*)
\end{align*}
\]
The definition for sequence uses set concatenation. The concatenation of two sets $K$ and $M$ is defined as: $KM = \{xy \mid x \in K, y \in M\}$. If for example $K = \{a, b\}$ and $M = \{c, d\}$, then $KM = \{ac, ad, bc, bd\}$, i.e. all combinations.

**Example** As an example of a language of a regular expression, we derive, from left to right, the language of the expression $((c|d)c)?$, which was the incorrect content model generated by XML-Spy and Oxygen in section 1:

$$L(((c|d)c)?)$$

$= \text{zero or more times}$

$$L((c|d)c) \cup \{\epsilon\}$$

$= \text{sequence}$

$$L(c|d)L(c) \cup \{\epsilon\}$$

$= \text{choice}$

$=(L(c) \cup L(d))L(c) \cup \{\epsilon\}$

$= \text{atom}$

$=(\{c\} \cup L(d))L(c) \cup \{\epsilon\}$

$= \text{atom}$

$=(\{c\} \cup \{d\})L(c) \cup \{\epsilon\}$

$= \text{set union}$

$\{c, d\} L(c) \cup \{\epsilon\}$

$= \text{atom}$

$\{c, d\} \{c\} \cup \{\epsilon\}$

$= \text{set concatenation}$

$\{cc, dc\} \cup \{\epsilon\}$

$= \text{set union}$

$\{cc, dc, \epsilon\}$

### 2.3 Rules

The choice operator ($|$) is associative, commutative, and idempotent. The concatenation operator is associative and the empty sequence $\epsilon$ is the unit of concatenation. In formulas this reads for all regular expressions $R$, $S$, and $T$ [12]:

$$R((S|T) = (R|S)|T \quad (2.1a)$$

$$R|S = S|R \quad (2.1b)$$

$$R|R = R \quad (2.1c)$$

$$R(ST) = (RS)T \quad (2.1d)$$

$$Re = R \quad (= \epsilon R) \quad (2.1e)$$
The rules for cardinalities are:

\[ R^* = \epsilon R R^* \]  
\[ R^+ = R R^* \]  
\[ R? = \epsilon R \]  

Other equations are:

\[ RR^* = R^* R \]  
\[ (R^* S^*)^* = (R|S)^* \]  
\[ (RS)^* R = R(SR)^* \]  
\[ XR|XS = X(R|S) \]  

The last rule says that concatenation distributes over choice, i.e. if three arbitrary expressions \( R, S, \) and \( X \) match the pattern on the left hand site, then we can replace the left hand by the right hand site. Using the distribution rule, \( ab|ac \) can be rewritten as \( a(b|c) \) and \( abc|abd \) as \( ab(c|d) \).

The distribution rule \( XR|XS = X(R|S) \) appears in another form, namely \( RX|SX = (R|S)X \). The first one is called left-factoring and the second one is called right-factoring. We will explain the use of the distribution rule (especially left-factoring) further in section 5.

All rules can be read from left to right and vice versa. Some rules have multiple occurrences due to commutativity (for example \( R? = \epsilon| R \) versus \( R? = R|\epsilon \)).

Whereas the empty string \( \epsilon \) often occurs, we will never use the empty set \( \emptyset \) in modeling XML element content. For completeness, we present the rules for the empty set:

\[ \emptyset| R = R = R|\emptyset \]  
\[ \emptyset R = \emptyset = \emptyset R \]  
\[ \emptyset^* = \epsilon \]

**Remark** Notice that some combinations of these rules lead to new useful ones. For example, the combination of the rules 2.1.e, 2.2.c, and 2.3.d leads to the following meaningful rule\(^1\):

\[ RX|X = R? X \]

**Remark** We have no evidence that this set of rules is complete.

**Remark** All of the rules listed can be proven using the definition of the language of regular expressions \( L \). We give an example:

**Proof** To be proven: \( R? = \epsilon| R \).

Proof by using the definition of \( L \): \( L(R?) = L(\epsilon| R) \)

\[ L(R?) = L(R|\epsilon) = L(\epsilon| R) \]

\(^1\)The derivation is as follows: \( RX|X \xrightarrow{2.1.e} RX|\epsilon X \xrightarrow{2.3.d} (R|\epsilon)X \xrightarrow{2.2.c} R? X \).
Rules 2.3.a, 2.3.b, and 2.3.c can be proven by induction too. As an example, we give a proof of rule 2.3.c.

**Proof** To be proven: $(RS)^{*}R = R(SR)^{*}$.

Proof by induction:

Base step: $(RS)^{0}R = R = R(SR)^{0}$

Assume: $(RS)^{k}R = R(SR)^{k}$ for all $k < n$, then we have to prove that $(RS)^{k+1}R = R(SR)^{k+1}$.

$(RS)^{k}R = RS(RS)^{k-1}R$, but $(RS)^{k-1}R = R(SR)^{k-1}$ by the induction hypothesis, thus:

$(RS)^{k}R = RS(RS)^{k-1}R = R(SR)(SR)^{k-1} = R(SR)^{k}$. □

3 XML element content and regular expressions

As we have seen, the content of an element definition specifies the order and number of occurrences of child elements. The DTD-notation for specifying element content is almost the same as the regular expression notation [16, 17]. Thus specifying element content we can use a variation on the regular expression notation:

- the alphabet consists of element names;
- concatenation is represented by comma (,) instead of using juxtaposition of its operands. As before, choice is represented by a vertical bar (|). Concatenation and choice behave as explained in section 2;
- the cardinality operators +, * and ? are represented and behave as explained in section 2;
- empty element content corresponds to the empty string $\epsilon$.

As with regular expressions, the operators ?, +, * bind stronger than concatenation, and concatenation binds stronger than the choice operator. Furthermore, only a restricted form of regular expressions called deterministic regular expressions is permitted. We will explain this type of expression in section 5.

An XML element can have empty content which is represented by the symbol $\epsilon$. As such, $\epsilon$ is a valid representation of XML element content. Because the DTD-language does not allow empty content as part of a composite content model, $\epsilon$ may not occur as a sub expression in a final model. For example $\epsilon|R$ must be rewritten in $R$.

The definitions of + and * need some attention in the context of XML. Both operators are often applied, because they can dramatically reduce the expression’s size. For example, $a|a, a|a, a$ can be rewritten to $a^{+}$. The price we have to pay for this reduction in size is a decrease in precision: the expression $a^{+}$ allows all sequences of one or more a’s, i.e. the language described by $a|a, a|a, a$ is a subset of the language described by $a^{+}$. To express this property, we introduce a partial ordering [9] between expressions: $R \leq S$ if $L(R) \subseteq L(S)$. And as
a consequence:

\[ \forall k : k \geq 1 : R^k \leq R^+ \]  
\[ \forall k : k \geq 0 : R^k \leq R^* \]  

where \( R^0 = \epsilon \) and \( R^{k+1} = R, R^k \).

**Remark** Actually, we need a new set of rules that fit the syntax of XML content models. However, we will use the rules from the previous section for both regular expressions as well as XML content models (the most important difference is the notation for concatenation: juxtaposition instead of comma).

Now we know how the DTD-notation of element content matches the notation for regular expressions, we can use the rules to rewrite regular expressions.

### 4 Rewriting element content

In this section we give three examples showing how we can rewrite element content using the rules from section 2 and 3. For now, we concentrate on conciseness of content models.

**Example** We take the content model of element book from our book example \( \text{title, author, author, author, chapter, chapter} \). Using the rewrite rules, we can rewrite this model as follows:

\[ \text{title, author, author, author, chapter, chapter} \leq (3.1.b) \text{title, author, chapter, chapter} \leq (3.1.b) \text{title, author, chapter}^+ \]

We have applied rule 3.1.b twice and have reached a concise form. In this rewriting we assume the number of authors and chapters may be one or more.

**Example** Assume the content of element recs: \( a, b, c|b, c|b, a, c|a, b, c, d \). This model can be rewritten as follows:

\[ a, b, c|b, c|b, a, c|a, b, c, d = (2.1.b \text{ twice}) \]
\[ b, c|b, a, c|a, b, c|a, b, c, d = (2.3.d) \]
\[ b, (c|a, c)|a, b, c|a, b, c, d = (2.5.a) \]
\[ b, a?, c|a, b, c|a, b, c, d = (2.5.a) \]
\[ b, a?, c|a, b, c, d? \]
Notice that this rewriting has been shortened using the derived rule \( RX \mid X = R^?X \) (2.5.a). Furthermore, in this case other rewritings are possible with other final models.

**Example** Consider \( a, (b,a)^*, b((a,b)^*) \). This model can be rewritten as follows:

\[
\begin{align*}
  a, (b,a)^*, b((a,b)^*) & = (2.3.c) \\
  a, b, (a,b)^*|(a,b)^* & = (2.2.a) \\
  a, b, (a,b)^*|a, b, (a,b)^*|\epsilon & = (2.1.c) \\
  a, b, (a,b)^*|\epsilon & = (2.2.a) \\
  (a,b)^* & \\
\end{align*}
\]

In this rewriting we have introduced the empty element as part of a composite content model in the second step. As stated earlier, element content can’t contain an empty element as sub expression. We always have to remove this symbol before ending the rewriting. Furthermore, this rewriting shows that sometimes an (intermediate) expression has to ‘grow’, i.e. the number of symbols increases before further simplification can happen. Application of rule 2.3.b. expands \( a,b,(a,b)^*|(a,b)^* \) into \( a,b,(a,b)^*|a,b,(a,b)^*|\epsilon \). After this expansion, we are able to reduce the expression to the simple form \( (a,b)^* \). In fairness, this example is more or less exotic in the context of XML!

The sequence of rule applications 2.2.a, 2.1.c, and 2.2.a in example 3 introduces another usable rule, namely:

\[
RR^*|R^* = R^+|R^* = R^* \quad (4.1a)
\]

Using this rule the rewriting reduces from four to two steps.

## 5 Deterministic content models

Now we know the rules and how to use them, we can introduce another important topic we will use. The W3C has decided that XML should be compatible with SGML. As a consequence, content models of DTD’s should be deterministic too. A content model is *deterministic* if for each element part of an instance XML-document, the belonging element in the content model can be determined without looking forward.

**Example** The content model \( (a,b)|(a,c) \) is not deterministic, because if an XML-parser reaches an element \( a \), the parser can not conclude, without looking forward, which \( a \) should be taken in the content model. If the next element is
a, b, the first one \((a, b)\) should be taken, if the next one is \(a, c\) the second one \((a, c)\) should be taken.

Formally, let \(\bar{r}\) stand for the expression obtained from the regular expression \(r\) by replacing the \(i\)th occurrence of symbol \(\sigma\) in \(r\) by \(\sigma_i\), for every \(i\) and \(\sigma\). For example, for \(r = b^*a(b^*a)^*\) we have \(\bar{r} = b_1^*a_1(b_2^*a_2)^*\).

**Definition** A regular expression \(r\) is deterministic if there are no words \(w\sigma_i v\) and \(w\sigma_j v'\) in \(L(\bar{r})\) such that \(i \neq j\) [3].

This definition is declarative. A functional one, which we will use for determining whether or not a content model is deterministic, is as follows:

**Definition** Generally, there are two situations in which non determinism occurs [19]:

1. If the content model contains \(X|Y\) and the sets of atoms that can start a string generated by \(X\) and \(Y\) are not disjoint. An example of this situation is \((a, b)|(a, c)\), in which case the set of starters generated by \((a, b)\) is \(\{a\}\), and the set of starters generated by \((a, c)\) is \(\{a\}\) too.

2. If the content model contains \(X^*, X^+, \) and/or \(X?\), and the set of atoms that can start a string generated by \(X\) is not disjoint from the set of atoms that can follow in this particular context. An example of this situation is \(a^*, a\), in which case the set of starters generated by \(a^*\) equals the set of starters generated by \(a\), namely \(\{a\}\).

**Removing non determinism** In the first situation of the functional definition, non determinism can always be removed by *left factorization*, using the rules 2.3.d or 2.5.a.. Left factorization is a term from grammar transformation theory [10]. Application of rule 2.3.d on \((a, b)|(a, c)\) results in: \(a, (b|c)\). Now, both content models are semantically equal but the second one is deterministic, i.e. the parser does not have to look one element forwards knowing which content model should be chosen.

Sometimes, preparatory steps are needed to isolate the part that has to be factored. The next example shows such a situation.

**Example** As an example, the content model \((a?, b)|(b, a?)\) is not deterministic.
This model can be turned into a deterministic one, as follows:

\[
(a?, b)|(b, a?)
= (2.5.a twice)
\]
\[
b|(a, b)|b|(b, a)
= (2.1.b)
\]
\[
(a, b)|b|b|(b, a)
= (2.1.c)
\]
\[
(a, b)|b|(b, a)
= (2.5.a)
\]
\[
(a, b)|(b, a?)
\]

By firstly applying rule 2.5.a twice, we turn the expression into case one of the functional definition: if the parser reads a 'b', it is unclear which model should be chosen. The first three steps prepare the model for factorization. The non-determinism is removed by factorization in the last step.

The second situation of the functional definition is more complicated. In the case of \(a^*, a\), equivalent deterministic expressions are \(a, a^*\) (application of rule 2.3.a) and \(a^+\) (application of the rules 2.3.a and 2.2.b). We give some extra examples:

**Example** Consider \(a|(a, b)^+\):

\[
a|(a, b)^+
= (2.2.b)
\]
\[
a|a, b, (a, b)^*
= (2.3.d)
\]
\[
a, (c|b, (a, b)^*)
= (2.2.c)
\]
\[
a, (b, (a, b)^*?)
\]

**Example** Consider \((a, b)^*, a\):

\[
(a, b)^*, a
= (2.3.c)
\]
\[
a, (b, a)^*
\]
Example Consider \((a,b)^*|a:\)

\[
(a,b)^*|a \\
= (2.2.a) \\
a,b, (a,b)^*|a \\
= (2.1.b) \\
\epsilon|a, b, (a,b)^* \\
= (2.3.d) \\
\epsilon|a, (\epsilon|b, (a,b)^*) \\
= (2.2.c) \\
\epsilon|a, (b, (a,b)^*)? \\
= (2.2.c) \\
(a, (b, (a,b)^*)?)
\]

Example Consider \((a,b)^*|(a,c)^*:\)

\[
(a,b)^*|(a,c)^* \\
= (2.3.b twice) \\
a,b, (a,b)^*|c|a, (a,c)^*|c \\
= (2.1.c and 2.1.b) \\
\epsilon|a, b, (a,b)^*|a, c, (a,c)^* \\
= (2.3.d) \\
\epsilon|a, (b, (a,b)^*)|c, (a,c)^* \\
= (2.2.c) \\
(a, (b, (a,b)^*)|c, (a,c)^*))?
\]

Although most non deterministic regular repressions can be made deterministic, there are non deterministic regular expressions that can not be rewritten into an equivalent deterministic one \([3, 6, 7]\). Examples of expressions that can’t be made deterministic are:

- \((a,b)^*, (a,c)^*\)
- \((a,b)^*, a?\)
- \((a,b)^*, (a|b)?\)

Remark After introduction of the \(\ast\) operator, the subexpression enclosed by this operator could be part of non determinism. Looking at the four examples concerned this situation, a useful strategy removing this non determinism might be:

1. Expand expressions of the form \(R^+, R^*,\) or \(R?\) using rules 2.2.a, 2.2.b, 2.2.c, and 2.3.b;
2. Rearranging parts using rules 2.1.b, 2.3.a, and 2.3.c;

3. Apply factorization using rules 2.3.d and 2.5.a.

Some expressions can’t be made deterministic. We see two possibilities to solve this problem:

- Make the expression less precise, i.e. enlarge the language the regular expression generates. For example, the expression \((a, b)^*, (a, c)^*\) can be rewritten into \((a, (b|c))^*\), which is deterministic. The price we have to pay for this solution is that \(L((a, (b|c))^*) \supseteq L((a, b)^*, (a, c)^*)\).

- Introduce extra levels in the XML-tree. If we have for example the content model \((a, b)^*, (a, c)^*\) as in:

```xml
<!ELEMENT abcs ((a,b)*,(a,c)*)>
```

we could remove the non determinism by introducing one extra level:

```xml
<!ELEMENT abacs (abs, acs)>
<!ELEMENT abs (a,b)*>  
<!ELEMENT acs (a,c)*>  
```

### 6 Systematically modeling element content

#### 6.1 Rewriting XML element content

For the systematical rewriting of XML element content we need a rewrite system extended with a normal form to strive for and a strategy to efficiently reach that normal form. A rewrite system is a set of rewrite rules [18] which are applicable on a particular domain.

**Domain** A domain describes which expressions are allowed and which are not. In case of regular expressions or XML element content, the domain is described in section 2.1 and 3.

**Rewrite rules** In a rewrite system a domain has a set of rewrite rules with which terms of the domain can be manipulated. Each rewrite rule consists of a left and a right hand side. Section 2.3 lists the rules which might be applied on regular expressions. Note that all the rules can be applied from left to right and vice versa. As we have shown in section 4, if for example the left hand side of a rule matches a (sub)expression, the matching part is replaced by the instantiated right hand side of the rule.
Normal form  A normal form (or standard or canonical form) of an expression in the domain of a term-rewriting system is an expression which has a certain form and cannot be rewritten anymore. Known examples in the field of logic are the disjunctive normal form (informally \((\ldots \land \ldots \land \ldots) \lor \ldots \lor (\ldots \land \ldots))\)], and the conjunctive normal form (informally \((\ldots \lor \ldots \lor \ldots) \land \ldots \land (\ldots \lor \ldots))\)]. Known normal forms of regular expressions are the star normal form (snf) and the epsilon normal form (enf) [6]. These types of normal forms are not useful in case of simplifying XML content models. In section 6.3 we will introduce a normal form we can use in modeling XML element content.

Strategy  A strategy can be regarded as a description of how the rewrite rules may be combined, i.e. in which order they may be applied, in order to effectively reach a normal form. In section 6.4 we will describe a strategy for reaching a normal form for XML element content.

Rewriting XML element content  To develop a system for the systematical rewriting of XML element content, we have to solve two problems. The first is a lack of an appropriate existing normal form of regular expressions for XML element content. For this, we describe a normal form we can use for modeling XML element content in section 6.3. The second one is that of a strategy. As we will explain, for a certain category of XML content models we are able to give a strategy. For all other models, we describe an approach in terms of rules of thumb.

Besides a normal form, we introduce some other aspects which are of importance in modeling XML element content. These are the correctness, precision, and conciseness of a content model.

6.2 Aspects of XML content models

6.2.1 Precision

A precise content model contains exactly those sequences of child elements that we want to have and no other one.

Definition  If \(R\) is a content model and \(T\) represents all sequences of child elements that we want to have, then \(R\) is a precise model if and only if \(L(R) = T\) holds.

It is very easy to obtain a precise model. The only thing we have to do is to write down the element content as a number of choices based on the example XML-file(s).

Example  If we have for example the following XML-file:

\[
<\text{recs}>
<\text{rec}>
<\text{a}/> <\text{b}/> <\text{a}/> <\text{b}/> <\text{a}/> <\text{b}/>
\]

15
a precise content model of element rec is: \(a, b, a, b, a, b|a, b|\epsilon|a, b, a, b\).

We call such a first precise content model the starting form (SF).

**Definition** The starting form (SF) of an XML content model is an expression consisting of a number of choices \(expr_1|expr_2|...|expr_n\), where each \(expr_i\) is a sequence of child elements of the element considered in the XML instance document(s).

By rewriting an SF, we can transform the model into another form and make the model more concise.

The precision of a content model cannot be tested using a validating parser: we have to check that only sequences of child elements we want are part of the sequences the content model encloses. The only approach here is to derive the language of the content model (see section 2.2) and check whether the sequences of child elements we want to have are the same as the language of the content model.

### 6.2.2 Correctness

A correct content model contains at least all sequences of child elements that we want to have.

**Definition** If \(R\) is a content model and \(T\) represents all sequences of child elements we want to have, then \(R\) is a correct model if and only if \(T\) is a subset of the language generated by \(R\): \(L(R) \supseteq T\).

**Example** The content model \((ab)^*\) encloses the model \(a, b, a, b, a, b|a, b|\epsilon|a, b, a, b\) and is as such a correct model.

The content model ANY is always a correct model.

**Definition** The form \(EF\) of an XML content model is a number of choices \(atom_1|atom_2|...|atom_n\), where each \(atom_i\) is an element name that can occur as child element of the element considered in the XML instance document(s).

Another always correct content model is \(EF^*\).

**Example** The atoms of the model \(a, b, a, b, a, b|a, b|\epsilon|a, b, a, b\) are a and b. As such, another correct model is \((a|b)^*\). Notice that \(L(\text{ANY}) \supseteq L((a|b)^*) \supseteq L((a, b)^*)\).
The use of the cardinality operators ∗ and + needs some attention in this matter, because they will introduce imprecise models. Assume the content model title, author, author, author, chapter, chapter. The operator + may only be used if one or more authors and/or chapters are allowed. If, in this case, exactly three authors and two chapters are required, the operator + may not be used since \( L(\text{author}, \text{author}, \text{author}) \subseteq L(\text{author}^+). \) The same goes for ∗. Whether the operators + and ∗ can be used depends on the context.

Note that XML Schema has the attributes minOccurs and maxOccurs to make precise definitions for such situations. These definitions can be expressed using regular expressions with numeric occurrence indicators (#res) [13], which is an extension of traditional regular expressions. They allow to define the number of required and allowed iterations of sub-expressions with numeric parameters. An expression \( R^{m..n} \) denotes the catenation of \( R \) with itself at least \( m \) and at most \( n \) times.

**Example** As a second example, the model \((c|d), c)? \) generated by XML-Spy is not a correct model (see section 1), because \( L(((c|d), c)?) = \{\epsilon, cc, dc\} \) and \( \{\epsilon, cc, dc\} \nsubseteq \{\epsilon, c, dc\} \).

The correctness of a content model can be tested using a validating parser: it checks all sequences of child elements in the example XML-document against the content model.

### 6.2.3 Conciseness

We define conciseness in terms of the size of an expression. The expression size is defined as the sum of all symbols, excluding brackets, where we assume brackets are used sparingly. The assumption is that expressions with a minimal number of symbols show the structure of the model most clearly. For example \((a|b)^*\) is equal to \((a^*, b^*)^*\), but the first one expresses the structure of the model more clearly: a sequence \((^*)\) of a’s and b’s in varying order \((|)\).

**Definition** The size of an expression is inductively defined by function size which takes a regular expression as argument and returns the number of symbols and operators of that expression:

\[
\begin{align*}
\text{size} \text{ atom} & = 1 \\
\text{size} (R|S) & = \text{size} R + 1 + \text{size} S \\
\text{size} (R, S) & = \text{size} R + 1 + \text{size} S \\
\text{size} R? & = 1 + \text{size} R \\
\text{size} R^* & = 1 + \text{size} R \\
\text{size} R^+ & = 1 + \text{size} R
\end{align*}
\]

**Remark** Notice that a case for empty content \((\epsilon)\) is not included, because empty content can not be part of an XML content model.
Example When we calculate the size of $author^+, title^+$ stepwise, we get:

\[
\begin{align*}
\text{size}(author^+, title^+) &= 3\text{-th case} \\
&= size(author^+) + 1 + size(title^+) \\
&= 6\text{-th case twice} \\
&= 1 + size(author) + 1 + 1 + size(title) \\
&= \text{first case twice} \\
&= 1 + 1 + 1 + 1 + 1 \\
&= \text{arithmetic plus} \\
&= 5
\end{align*}
\]

Remark In cases of correct models, the content model $EF^*$ is always the most concise form but often not strict enough. Therefore, conciseness alone should not be a criterion, but should be evaluated in combination with the level of imprecision.

6.3 A normal form for XML content models

Assuming we start with an XML content model in SF, our first model is a number of choices:

\[
\text{expr}_1 | \text{expr}_2 | \ldots | \text{expr}_n
\]

where each $\text{expr}_i$ is a particular sequence of child elements of the element considered in the XML instance document(s) (see section 6.2.1). The only requirement we further ask for is determinism (see section 5).

Definition A normal form of an XML content model is a starting form (SF) which is deterministic. If we express the property that an expression $\text{expr}$ is deterministic as $\text{isDet}(\text{expr})$, then a normal form for XML element content (XNF) is:

\[
\text{isDet}(SF) \quad (6.1)
\]

Notice that the property of determinism includes that all double occurrences are removed (application of rule $R|R = R$ (2.1.c)). Additional, because the DTD-language does not allow empty content as part of a composite content model, $\epsilon$-signs must be removed (application of rule $R? = \epsilon|R$ (2.2.c). In section 7 we will prove that every XML content model in SF can be made deterministic.

Example Assuming the content model of element rec in the XML-file recs (see section 6.2.1), our first model equals:

\[
a, b, a, b, a, b|a, b|\epsilon|a, b, a, b
\]

Because this model is not deterministic, we rewrite the model into a determin-
istic one:

\[
\begin{align*}
  a, b, a, b, a, b | a, b | e | a, b, a, b \\
  = (2.1.b \text{ twice}) \\
  e | a, b | a, b, a, b, a, b, a, b \\
  = (2.5.a) \\
  e | a, b | a, b, (a, b)? \\
  = (2.5.a) \\
  e | a, b, (a, b, (a, b)?)? \\
  = (2.2.c) \\
  (a, b, (a, b, (a, b)?))? 
\end{align*}
\]

### 6.4 Strategies

We assume we start with an XML content model in SF. In the case of a precise model, we can give a strategy for transforming an SF into a deterministic precise model. In the case of a correct model, we are not able to give a strategy. Instead, we present some rules of thumb.

#### 6.4.1 A strategy for Precise Models

We start with an XML content model in SF based on one or more XML instance documents. Firstly, we remove redundant choices of the form \( R|R \) by applying rule \( R|R = R \). On the resulting model, we search iteratively under top-down control for subexpressions of the form \( X|Y \) of which the sets of starters of \( X \) and \( Y \) are not disjoined. If we find such a subexpression, we apply one of the factorization rules \( XR|XS = X(R|S) \) and \( XR|X = XR? \) on this expression, eventually after application of the commutativity rule \( R|S = S|R \). This part of the procedure stops when for all subexpression of the form \( X|Y \) the sets of starters of \( X \) and \( Y \) are disjoined. Because the SF started with can contain empty content and empty content can be introduced by application of rule \( XR|XS = X(R|S) \), we end with removing \( e \) signs by applying rule \( e|R = R? \). The result of this strategy is a content model in XNF. All rules used, must be applied from left to right. More formally:

**Input:** a content model \( expr \) in SF

Remove redundant choices of the form \( R|R \) by applying rule \( R|R = R \).
Repeat until \( isDet(expr) \)

- Search under top down control for a subexpression of the form \( X|Y \)
  of which the sets of starters of \( X \) and \( Y \) are not disjoined.
- If needed, rearrange using the commutativity rule: \( R|S = S|R \).
- Factorize this choice using one of the rules \( XR|XS = X(R|S) \)
  or \( XR|X = XR? \) from left to right.
- If needed, remove empty content by applying rule \( e|R = R? \) from left to right

**Output:** the content model \( expr \) in XNF.
After applying this strategy on a content model in SF, the resulting expression is deterministic and the size of the resulting model is less or equals the size of the input model (see section 7).

For determining the set of starters of an expression, we will define a function look-ahead in section 7.

Remark The applied rewrite policy, searching and rewriting iteratively under top-down control, is also known as outermost reduction, where each step reduces an outermost redex (a reducible expression) [5].

Example Assume as an SF the following content model: $a, b|a, b, a, b|c|c, c|a, b, c$. We will develop a precise model in XNF:

$$a, b|a, b, a, b|c|c, c|a, b, c$$

= (rearrange using 2.1.b)

$$\epsilon|a, b|a, b, a, b|a, b, c|c|c, c$$

= (2.5.a)

$$\epsilon|a, b(a, b)?|a, b, c|c|c, c$$

= (2.3.d)

$$\epsilon|a, b, ((a, b)?|c)|c|c, c$$

= (2.5.a)

$$\epsilon|a, b, ((a, b)?|c)|c, c?$$

= (2.2.c)

$$(a, b, ((a, b)?|c)|c, c?)?$$

This model is precise and in XNF.

Example Assume as an SF the following content model: $a|a, x|a, x, b$. We will develop a precise model in XNF:

$$a|a, x|a, x, b$$

= (2.5.a)

$$a, x?|a, x, b$$

= (2.3.d)

$$a, (x?|x, b)$$

= (2.2.c)

$$a, (\epsilon|x|x, b)$$

= (2.5.a)

$$a, (\epsilon|x, b?)$$

= (2.2.c)

$$a, (x, b?)?$$
This model is precise and in XNF. This derivation looks like long for this model. The reason is the top-down approach. An alternative rewriting is as follows:

\[
\begin{align*}
\text{a | a, x | a, x, b} \\
&= (2.3.d) \\
\text{a | a, (x | x, b)} \\
&= (2.5.a) \\
\text{a | a, (x, b?)} \\
&= (2.5.a) \\
\text{a, (x, b?)?}
\end{align*}
\]

### 6.4.2 Rules of thumb for Correct Models

Again, we start with an XML content model in SF based on one or more XML instance documents. During rewriting, it is allowed to introduce the cardinality operators \( \ast \) and +. Now, we have to deal with two problems:

- Introduction of these cardinality operators can introduce non determinism. As we have mentioned in section 5, this type of non determinism can’t always be removed by rewriting the expression. As we have stated, we could solve this situation by make the expression less precise or by the introduction of extra levels in the XML-tree.

- Whereas we have a strategy for precise models guaranteeing a content model in XNF, for correct models we are unable to develop such a strategy which is valuable. In section 6.2.2, we have seen that the model EF\( \ast \) is always a correct model. But, most times, this model is not strict enough. The problem is that a strategy for correct models can not automatically decide between precision and conciseness. Furthermore, whether the \( \ast \) operator can be introduced, depends on the context. For both problems, human intervention is needed. As a result, in stead of a strategy we give some rules of thumb.

**When to introduce the operators \( \ast \) and +?** There are two when-questions in introducing these operators:

1. **Do we introduce the operators \( \ast \) and + before or after factorization?** Generally, factorization removes repetitive parts of an expression and as a result reduces the possibility to introduce the operators \( \ast \) and +. If we have for example the model \( a, b | a, b, a, b | b \), factorization results in the model \( a, b, (a, b)? | b \). On the other hand, if we assume the subexpression \( a, b \) is repetitive and we first introduce the operator +, the resulting model could be \( (a, b)^+ | b \). We recommend introduction of the operators \( \ast \) and/or + before factorization take place.

2. **Are there semantic constrains for introducing the operators \( \ast \) and +?** The answer is certainly yes. A book consists of one or more chapters, so a
content model of $chapter^+$ is reasonable. But a model as $isbn^*, chapter^+$ is incorrect, because each book has exactly one isbn (International Standard Book Number). This question points to the fact that whereas modeling XML-content can be supported by computer programs, introduction of cardinality operators is man’s handwork.

Rules of thumb  Again, we assume as input a content model in SF.

Step 1. Remove redundant choices of the form $R|R$ by applying rule $R|R = R$.

Step 2. Introduce the operators $*$ and/or $+$. We distinguish three situations:

i) After introduction of the operator $*$ or $+$, the subexpression enclosed by the operator is not part of non determinism. For example, the subexpression $a,b|a,b,a,b$ in $a,b|a,b,a,b$ can be replaced by $(a,b)^+$, resulting in the model $(a,b)^+|b$. The subexpression $(a,b)^+$ is not part of non determinism. Our experience shows that this situation is most frequent.

ii) After introduction of the operator $*$ or $+$, the subexpression enclosed by the operator is part of non determinism, but the non determinism can be removed by rewriting the expression. For example, the subexpression $a,b|a,b,a,b$ in $a,b|a,b,a,b$ can be replaced by $(a,b)^+$ resulting in $(a,b)^+|a,b,c$. The subexpression $(a,b)^+$ is part of non determinism. This non determinism can be removed by (see section 5):

1) Expand expressions of the form $R^*$, $R^+$, or $R?$ using rules 2.2.a, 2.2.b, 2.2.c, and 2.3.b;
2) If needed, rearranging parts using rules 2.1.b, 2.3.a, and 2.3.c;
3) Apply factorization using rules 2.3.d and 2.5.a.

Applying this approach to our example results in:

\[(a,b)^+|a,b,c\]
\[= (2.2.b)\]
\[a,b,(a,b)^+|a,b,c\]
\[= (2.3.d)\]
\[(a,b,((a,b)^+|c))\]

In many cases, the non determinism introduced can be removed by this approach.

iii) After introduction of the operator $*$ or $+$, the subexpression enclosed by the operator is part of non determinism, but the non determinism can not be removed by rewriting the expression. As stated in section 5, there are two possibilities to solve such a situation:
a) Make the expression less precise, i.e. enlarge the language the regular expression expresses. For example, the expression \((a,b)^*,(a,c)^*\) can be rewritten into \((a,(b|c))\)^*, which expression is deterministic.

b) Introduce extra levels in the XML-tree. If we have for example the content model \((a,b)^*,(a,c)^*\) as in:

```xml
<!ELEMENT abcs ((a,b)*,(a,c)*)>
```

we could remove the non determinism by introducing one extra level:

```xml
<!ELEMENT abacs (abs, acs)>  
<!ELEMENT abs (a,b)*>  
<!ELEMENT acs (a,c)*>
```

Step 3. Optimize the expression by applying factorization and removing \(\epsilon\) signs.

**Remark** In almost all cases, we can start with a content model in SF which summing up exactly all sequences of child elements of the element considered. Sometimes however, we can not sum up exactly these sequences. An example of such a situation is a content model representing the moves of chess game: \((white, black)^+, white\). We know the moves will alternate white and black, but the number of moves and which color will be the last one are uncertain. In such cases, we start directly with step 2.

**Example** Again, assume as an SF the content model: \(a|b|a, b|a, b|\epsilon|c|c, c|a, b, c\).

We will develop a correct model in XNF. Now, the cardinality operators \(*\) and \(+\) are allowed.

\[
a, b|a, b, a, b|\epsilon|c|c, c|a, b, c
= \text{(rearrange using 2.1.b)}
\]

\[
\epsilon|a, b|a, b, a, b|a, b, c|c|c, c
\leq \text{(introduce * operator)}
\]

\[
(a, b)^*|a, b, c|c, c
= \text{(2.2.a)}
\]

\[
\epsilon|a, b, (a, b)^*|a, b, c|c|c, c
= \text{(2.3.d)}
\]

\[
\epsilon|a, b, ((a, b)^*|c)|c|c, c
= \text{(rearrange using 2.1.b)}
\]

\[
a, b, ((a, b)^*|c)|c|c, c
\leq \text{(introduce * operator)}
\]

\[
a, b, ((a, b)^*|c)|c^*
\]

This model is correct and in XNF. If we assume all numbers of ab’s followed by all numbers of c’s are allowed, we get:

\[(a, b)^*, c^*\]
This model is correct and in XNF too. The second one is more concise then the first one. But the second one is less precise. Often, conciseness is at the expense of precision.

**Remark** The methods described for modeling precise and correct models, are applicable in cases where we have one XML-file with one of more example records, a number of XML files, and even without an example XML-file at all. In all these cases, we are able to sum up, by reading and/or thinking, all sequences of child elements of each element we want to have, i.e. to draw up a content model in SF. In a second phase, we can rewrite this form in precise or correct XNF.

## 7 Formalization

### 8 Algorithmic approach

Besides rewriting regular expressions as explained in the previous section, there are algorithmic solutions for simplifying regular expressions. For the sake of completeness, we briefly mention this approach in this section. More information can be found in [11, 10].

These algorithms are not based on rewrite rules. Instead, these algorithms transform a regular expression into a Finite-state Automation (FA). There are two types of FA’s, namely Non-deterministic (NFA) and Deterministic ones (DFA). If the transformation of a regular expression results in an NFA, this NFA should be firstly transformed into a DFA. Turning an NFA into a DFA usually increases the size of the automation by some factor. Often, a DFA can be simplified by applying a DFA minimization algorithm. Finally, the optimized DFA is transformed back into a regular expression (however, the last transformation, from DFA to a deterministic regular expression, is not always possible).

Another approach uses a probabilistic algorithm that learns content models, and selects the one that best describes the samples [3].

## 9 Conclusion

In these notes, a method is introduced for modeling XML-content models systematically. The method is based on rewriting regular expressions, which XML content models actually are. The method is introduced for DTD-elements, but will be useful for modeling XML-Schema complex types with complex content too.

**Acknowledgements**

The author thanks Bastiaan Heeren, Arnold van der Leer, and Lex Bijlsma for reading and commenting on draft versions of these notes.
References


