Modeling XML Content Explained

Harrie Passier

Bastiaan Heeren

Technical Report UU-CS-2011-019
June 2011

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Modeling XML Content Explained

Harrie Passier
School of Computer Science
Open Universiteit Nederland
Harrie.Passier@ou.nl

Bastiaan Heeren
School of Computer Science
Open Universiteit Nederland
Bastiaan.Heeren@ou.nl

ABSTRACT

Over the years, a lot of course material has been developed to explain to undergraduate students the fundamentals of XML, and schema languages such as DTD and XML-Schema. Typically, the syntax of these languages is discussed and examples are given. How to find a schema for some XML content is often not covered by the material. As a result, students have problems to start with modeling a complex schema, many of their inferred XML schemas are too liberal, and some are even incorrect. In this paper we present a systematic approach for modeling XML content models based on rewriting regular expressions. A small-scale experiment has demonstrated that the quality of the models is improved, and that the approach helps students to begin modeling XML content.

Categories and Subject Descriptors

F.4.3 [**Mathematical and formal languages**]: Formal Languages; K.3.2 [**Computers and education**]: Computer and Information Science Education

General Terms

Design, Experimentation, Languages, Measurement

1. INTRODUCTION

Many of today's computer science courses introduce and explain their topics without mentioning their underlying formal methods [9]. As a result, it remains unclear how to construct a program, an algorithm, a data structure, et cetera. For example, introductory courses on data structures and algorithms are often limited to the common data structures and accompanying algorithms, and how to use these, but how to develop algorithms on new data structures is not always explained. Instead, teaching methods and engineering approaches are used that mainly rely on inspiration and intuition, and this does not always work out well. As a result, students are often not sufficiently aware of what to do and why: they need and ask for more guidance in terms of "how to do" a particular task.

In this paper, we discuss the use of formal methods in an introductory course on XML at bachelor level. We focus on XML schema languages, such as the Document Type Definition (DTD) language and XML-Schema. More specifically, we present an approach based on regular expressions for deriving content models for XML elements in a systematic way. Consider the following XML instance document:

```
<book>
  <title>Touch of class</title>
  <author>Bertrand Meyer</author>
  <chapter>The industry of pure ideas</chapter>
  <chapter>Dealing with objects</chapter>
  <!-- Much more chapters follow -->
</book>
```

Based on this instance only, a `book` element contains a `title`, an `author`, and one or more `chapter`s. If we also use our common sense, then we could allow several authors for a book, but not multiple titles.

Although the similarities between schema languages and regular expressions are well understood, books and teaching material do not use this to their advantage. Typically, a number of (small) examples is given, but without an explanation of how the resulting content model was found. We are not aware of books on XML that introduce regular expressions to provide a deeper insight into content models, or a systematic way to model XML content.

We have made the following observations about students asked to give a suitable XML content model for some XML instance document:

- They have difficulties to get started: they need assistance with the first steps in constructing a complex content model, or the evaluation of such a model;
- The process of finding a model is not structured, and involves a lot of trial-and-error;
- Resulting models are too liberal (the schema accepts too many XML documents), or even incorrect (parts are missing in the schema);
- Resulting models are not deterministic;
- Students are unable to evaluate a generated model on correctness and precision.

Similar observations can be made about learning material for other XML-related languages, such as the navigation and selection language XPath, and the transformation languages XQuery and XSLT.

We present a systematic approach based on rewriting regular expressions [1] that helps students in constructing con-

Construct	DTD notation	RE notation
empty set		$\emptyset$
empty string	*EMPTY*	ϵ
alphabet	element names	atoms
sequence	R, S	RS
choice	$R \mid S$	$R \mid S$
zero or one	$R?$	$R?$
zero or more	R^*	R^*
one or more	R^+	R^+

Figure 1: Syntax of DTDs versus REs

tent models as part of a schema. By establishing a link between schema languages and regular expressions, it becomes much easier to reason about content models, and to manipulate these models. Rewrite rules on regular expressions pave the way for a stepwise derivation of a content model. We distinguish between precise content models (describing exactly the set of allowed sequences of XML elements, but nothing more), and correct models (describing at least the sequences of XML elements we want to have). For both cases we specify a strategy that makes precise how and where to apply the rewrite rules. This is the paper's first contribution.

We have tested our modeling approach on a group of students. The test clearly shows that the approach can help students in learning to model complex XML content models. Test and interview results are given in the last part of the paper. This is our second contribution.

The paper is structured as follows. Section 2 gives an introduction to DTDs and regular expressions, and presents rules to rewrite these expressions. Section 3 then explains how to make a content model deterministic, a requirement of the DTD language. Sections 4 and 5 define strategies for precise and correct content models, respectively. We then discuss our small-scale experiment (Section 6). The last sections give related work, draw conclusions, and give directions for future work.

2. DTDS AND REGULAR EXPRESSIONS

We start with a comparison of DTDs and regular expressions (REs), followed by a formal definition of the language that is generated by an RE. In the final part, we give some rewrite rules for manipulating expressions.

2.1 Syntax

A DTD lists all the elements that can be part of an XML document by means of element declarations, such as:

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

The content model of an XML element specifies which child elements may occur, and in which order. In our example, `(title, author+, chapter+)` is the content model of element `book`. Each element `book` must have a `title` element, followed by one or more `author` elements, and one or more `chapter` elements. In the remainder of this paper, we choose DTD as our schema language, but our approach works for other languages as well (such as XML-Schema).

The syntax of a content model differs slightly from standard RE notation: Figure 1 shows the correspondence between the notations. There is no counterpart of the empty set construct in DTD notation. Furthermore, ϵ concisely denotes the empty string. Also observe how the commas are dropped for sequences (since the atoms of an RE are generally assumed to be single characters, unless otherwise noted). For reasons of presentation, we adopt the RE syntax in this paper, without the $\emptyset$ construct.

$$
\begin{aligned}
R \mid (S \mid T) &= (R \mid S) \mid T && \text{(1a)}\\
R \mid S &= S \mid R && \text{(1b)}\\
R \mid R &= R && \text{(1c)}\\
R(ST) &= (RS)T && \text{(1d)}\\
\epsilon R &= R && \text{(1e)}\\
R\epsilon &= R && \text{(1f)}\\
R? &= \epsilon \mid R && \text{(2a)}\\
R^* &= \epsilon \mid RR^* && \text{(2b)}\\
R^+ &= RR^* && \text{(2c)}\\
RS \mid RT &= R(S \mid T) && \text{(3a)}\\
RT \mid ST &= (R \mid S)T && \text{(3b)}\\
R^*R &= RR^* && \text{(3c)}\\
(RS)^*R &= R(SR)^* && \text{(3d)}
\end{aligned}
$$

Figure 2: Rewrite rules on REs

We use R, S, and T to represent arbitrary REs, and $a, b, c, \ldots$ for the atoms in our examples. The standard precedence levels apply: the unary operators (?, *, and +) bind stronger than sequence, which binds stronger than choice. Parentheses are used to group expressions. Hence, the expression $ab^* \mid c$ is interpreted as $(a(b^*)) \mid c$, and not $a(b^* \mid c)$ or $(ab)^* \mid c$.

2.2 Language

An RE describes a possibly infinite set of sentences, which we call the language generated by that expression, denoted by $\mathcal{L}(\cdot)$. This can be defined inductively as follows [8]:

$$
\begin{aligned}
\mathcal{L}(\epsilon) &= \{\epsilon\}\\
\mathcal{L}(a) &= \{a\}\\
\mathcal{L}(ST) &= \mathcal{L}(S)\mathcal{L}(T)\\
\mathcal{L}(S \mid T) &= \mathcal{L}(S) \cup \mathcal{L}(T)\\
\mathcal{L}(S?) &= \mathcal{L}(\epsilon \mid S)\\
\mathcal{L}(S^*) &= (\mathcal{L}(S))^*\\
\mathcal{L}(S^+) &= \mathcal{L}(SS^*)
\end{aligned}
$$

Here, concatenation of two sets, written XY, is short-hand notation for $\{\, xy \mid x \in X, y \in Y \,\}$. The star-closure of a set, X^*, equals $X^0 \cup X^1 \cup \ldots$, where:

$$
\begin{aligned}
X^0 &= \{\epsilon\}\\
X^{n+1} &= XX^n
\end{aligned}
$$

Similarly, R^n is used as shorthand notation for a sequence of n occurrences of regular expression R.

These definitions provide the proper foundation to reason about and manipulate REs. In forthcoming sections, we use this to evaluate content models on correctness.

2.3 Rewrite rules

Figure 2 presents a list of rewrite rules that operate on REs. The first set of rules (1a–1f) expresses some basic properties of the choice and sequence operators: choice is associative, commutative, and idempotent, whereas sequence is associative and has ϵ as its unit. Soundness of these rules follows straightforwardly from the language generated for both sides of the equation. The rules for associativity

$$
\begin{array}{rcll}
R^n & \Rightarrow & R^* & (\textit{if } n \geq 0) \quad (4a) \\
R^n & \Rightarrow & R^+ & (\textit{if } n \geq 1) \quad (4b) \\
R^*S^* & \Rightarrow & (R \mid S)^* & (4c)
\end{array}
$$

Figure 3: Directed rewrite rules

(1a and 1d) are typically performed implicitly, and parentheses are dropped accordingly.

The second set of rules (2a–2c) defines a translation for each of the cardinality operators. These rules show that all occurrences of $R?$ and R^+ can be removed from an expression. On the other hand, R^* can be expanded one step.

The last set of rules is for making expressions deterministic, which is discussed in the next section. We have rules for left factoring (3a), and right factoring (3b). Rule 3d (and 3c as a special case) help in rearranging expressions involving R^* (under the right circumstances these can be shifted to the right). More rewrite rules can be added to the collection, for example by combining existing ones. When modeling XML content with REs, it is convenient to have a rich set of rules that covers common patterns. Note that the rules in Figure 2 can be applied in both directions (i.e., also from right to left) because both sides are equal.

When modeling XML content, one typically uses the cardinality operators to reduce the size of the model. For example, $a \mid aa \mid aaa$ can be written as a^+, which is far more concise. The price we pay for this reduction in size is a loss of precision: the latter expression now also accepts $aaaa$. Figure 3 shows two more rewrite rules for the introduction of cardinality operators. These rules are directed from left to right.

Semantically, these directed rules extend the language that is generated. To specify this property, we introduce a partial ordering between REs: $R \leq S$ if and only if $\mathcal{L}(R) \subseteq \mathcal{L}(S)$. A directed rewrite rule $R \Rightarrow S$ must satisfy $R \leq S$, and indeed, the rules of Figure 3 have this property.

3. REMOVING NON-DETERMINISM

XML is defined to be compatible with SGML, and as a consequence, content models of DTDs have to be deterministic. A content model is deterministic if an XML processor can check a document against a DTD without looking forward in the document (i.e., inspecting only the current element). Generally, there are two situations in which non-determinism occurs [13]:

1. A content model contains $R \mid S$ and the sets of element names that can start a sequence in $\mathcal{L}(R)$ and $\mathcal{L}(S)$ are not disjoint. For example, $ab \mid ac$ is not deterministic because the set of starters (known as the *first* set [4]) is $\{a\}$ for both alternatives.

2. A content model contains $R?$, R^*, or R^+, and the set of element names that can start a sequence in $\mathcal{L}(R)$ is not disjoint with the set of names that can follow in this particular context (the *follow* set [4]). An example of such a non-deterministic expression is $(ab)^*ac$.

3.1 Strategy for removing non-determinism

We now present a strategy for the stepwise removal of non-determinism: rewrite problematic subexpressions (one of the two situations described above) until we have reached a deterministic expression. We discuss the two situations.

Situation 1.

Given is a subexpression $R \mid S$ with at least one element that is starter of R and S. Let this element be a. The non-determinism can be removed in three steps.

(a) Rewrite R and S until element a is the first of a sequence. This involves expanding cardinality operators (2a–2c), removing ϵ in sequences (1e), and distributing sequence over choice (3b). Rules 1c, 3c, and 3d (and variants for the other operators) can provide a shortcut.
(b) If needed, rearrange alternatives (1b) so that the sequences starting with a are adjacent.
(c) Apply the factorization rule 3a. In some cases, an ϵ has to be introduced first (1f).

Example 1. Applying the above strategy to the non-deterministic expression $(ab)^+ \mid bc \mid abc$ results in:

$$
\begin{array}{rcll}
(ab)^+ \mid bc \mid abc & = & ab(ab)^* \mid bc \mid abc & (2c) \\
& = & ab(ab)^* \mid abc \mid bc & (1b) \\
& = & ab((ab)^* \mid c) \mid bc & (3a)
\end{array}
$$

Example 2. The top-level alternatives in the following example make the expression non-deterministic:

$$
\begin{array}{rcll}
(a \mid b)a \mid a & = & aa \mid ba \mid a & (3b) \\
& = & aa \mid a \mid ba & (1b) \\
& = & aa \mid a\epsilon \mid ba & (1f) \\
& = & a(a \mid \epsilon) \mid ba & (3a)
\end{array}
$$

The last two steps are a good candidate for adding a new, derived rewrite rule to the collection: $RS \mid R = R(S \mid \epsilon)$.

Situation 2.

The second case is more complicated because some REs cannot be transformed into a deterministic form [2, 5, 4]. Examples of such expressions are $(ab)^*(ac)^*$ and $(ab)^*a?$. Note that deterministic REs should not be confused with deterministic finite-state automata (DFA) [7], another well-known formalism. Every RE can be transformed into an equivalent DFA, and the other way around. An RE constructed from a DFA, however, is not automatically deterministic.

Suppose we have some subexpression $R?$, R^*, or R^+, which has element a as a starter. Furthermore, a is also in the follow set of the subexpression at hand. We proceed by case analysis on the operator used.

(a) For content model $R?$, we remove the operator by applying rule 2a, resulting in the alternative $\epsilon \mid R$. Then, this alternative should be combined with its context, for instance by using distribution (rule 3b, from right to left). Eventually, we arrive at a situation 1 problem. This case also covers non-deterministic expressions of the form $\epsilon \mid R$ (instead of $R?$) that have a non-disjoint follow set.
(b) At its best, removing non-determinism involving R^* can be done with rules 3c and 3d. In some cases, expression R or its context needs some rewriting before these rules are applicable. If this does not work (e.g. because it is impossible), then two possibilities remain to deal with the situation:
   - Make the expression less precise, and extend the language generated by the RE.
   - Introduce an extra level in the XML tree, and circumvent the ambiguity altogether.

(c) In case of R^+, we use rule 2c. This reduces the problem to the case for R^*.

Example 3. The following derivation illustrates the case for an optional part:

$$\begin{array}{rcll} (ab)?a & = & (\epsilon \mid ab)a & \text{(2a)} \\ & = & a \mid aba & \text{(3b and 1e)} \\ & = & a(\epsilon \mid ba) & \text{(3a and 1f)} \end{array}$$

Example 4. Consider the regular expression $(ab)^*(ac)^*$, for which there is no equivalent deterministic RE. We can opt for a less precise (but deterministic) model:

$$\begin{array}{rcll} (ab)^*(ac)^* & \leq & (ab \mid ac)^* & \text{(4c)} \\ & = & (a(b \mid c))^* & \text{(3a)} \end{array}$$

Alternatively, we can decide to introduce an extra level by defining new elements. Suppose that the model $(ab)^*(ac)^*$ belongs to an element `abacs`. We can split the model into two parts, resulting in the following element declarations:

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

The new element names are `abs` and `acs`. Notice that introducing extra levels is not related to rewriting REs.

3.2 Normal form for content models

Besides the requirement that content models have to be deterministic, an ϵ should not be part of a composite content model according to the DTD language specification. This means that in the end, ϵ's have to be removed, which is fairly simple (e.g., with rules 2a, 1e, and 1f). This gives us a normal form for content models.

Definition 1. A content model M is in XML normal form (XNF) if and only if M is deterministic, and no ϵ is present in M (except when M itself is ϵ).

Reaching XNF is not a goal of its own, but rather a final step after the strategies presented next (Section 4 and Section 5).

4. PRECISE CONTENT MODELS

We now turn our attention to deriving a content model from some instance document. A content model should not be made too liberal without careful thought: after all, schema languages are used in the first place to reject documents and to spot inconsistencies. We start by considering precise content models only: a precise content model contains exactly those sequences of child elements that we want to have, and no others.

Definition 2. Let M be a content model, and X a set of sequences of child elements. Then M is a precise model for X if and only if $\mathcal{L}(M) = X$.

Obtaining a precise model for some XML content is rather easy. First, we write down all sequences of child elements for some particular element that appear in the instance document. These sequences are the choices of the starting model. For example, suppose we have:

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

This instance document contains four sequences of child elements for element `rec`. Hence, the starting model is $ab \mid \epsilon \mid abab \mid ab$. We call such a first approximation of a precise content model the starting form (SF).

The next step is to rewrite the content model in starting form, and turn the model into XNF without loosing precision. A strategy for this step is discussed next.

4.1 Strategy for precise content models

The strategy described in Section 3 can turn any content model into XNF. However, if we start with a model in SF and look for a precise model, a much simpler strategy is sufficient. More specifically, cardinality operators are not present in the starting model, nor are they introduced during rewriting (since that would make the model no longer precise). The strategy for precise content models is as follows:

Input: An XML content model in SF.
Output: A precise XML content model in XNF.

Step 1. Remove redundant choices of the form $R \mid R$ by applying rule 1c. If the duplicate choices are not adjacent, then change the order (rule 1b).

Step 2. Remove situation 1 type of non-determinism (for subexpressions of the form $R|S$) by repeating the strategy of Section 3.1 until the model is deterministic.

Step 3. To reach XNF, we remove all occurrences of ϵ. For this, we apply rule 2a, from right to left.

The strategy for precise content models returns canonical models (up to the order of choices). The order in which the rules are applied does not influence the result. Removing duplicates early (step 1) helps to shorten the derivations. Also note that the size of the final model is never larger than the original model.

Example 5. We continue with the starting form for element `rec`, introduced earlier in this section. We rewrite its model into a precise model in XNF.

$$\begin{array}{rcll} ab \mid \epsilon \mid abab \mid ab & = & ab \mid ab \mid \epsilon \mid abab & \text{(1b)} \\ & = & ab \mid \epsilon \mid abab & \text{(1c)} \\ & = & \epsilon \mid ab \mid abab & \text{(1b)} \\ & = & \epsilon \mid ab\epsilon \mid abab & \text{(1f)} \\ & = & \epsilon \mid ab(\epsilon \mid ab) & \text{(3a)} \\ & = & \epsilon \mid ab(ab)? & \text{(2a)} \\ & = & (ab(ab)?)? & \text{(2a)} \end{array}$$

4.2 Using the strategy

The strategy for precise models should be used if the number of choices in the SF is relatively small, and rewriting the model in a precise way is still manageable. Also use the strategy if a precise model is needed, i.e., the context demands a content model representing exactly the sequences of child elements in the SF and no other ones.

Precise models are not always desirable in practice, however, because models quickly become too verbose. For example, a book record has exactly one ISBN, but multiple authors and chapters. In this case, a content model with cardinality operators $(isbn, author^+, chapter^+)$ is more appropriate than a model without these operators.

The precision of a content model cannot be tested with a validating parser. The only approach here is to manually generate 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.

5. CORRECT CONTENT MODELS

A correct content model contains at least the sequences of child elements we want to have, and possibly more.

Definition 3. Let M be a content model, and X a set of sequences of child elements. Then M is a correct model for X if and only if $\mathcal{L}(M) \supseteq X$.

For instance, $(ab)^*$ is a correct content model for $\{\epsilon, ab, abab\}$, but not a precise model since $ababab \in \mathcal{L}((ab)^*)$. Correct models are generally more concise than precise models: the trade-off is that they can be more liberal than needed.

Smaller models show the structure more clearly. For example, $(a \mid b)^*$ is equal to $(a^*b^*)^*$, but the first one is (arguably) simpler. Minimizing the size of an expression should not be the only goal though. A model that allows everything (e.g. $(a_1 \mid a_2 \mid \ldots \mid a_n)^*$ where $a_1 \ldots a_n$ are all existing elements, which can be abbreviated to ANY) is concise, but defeats the purpose of writing content models. The challenge is to find the right balance between conciseness and precision. For this, expert knowledge about the domain being modeled is needed. For example, $chapter^+$ is reasonable for a book record, whereas $isbn^*$ is questionable. Such decisions cannot be made automatically by a strategy.

5.1 Strategy for correct content models

We now present a strategy for correct (but not necessarily precise) content models. This strategy introduces cardinality operators during rewriting. As a rule of thumb, cardinality operators should be introduced early on, and before factorization, because the initial model in SF best exposes the replicated parts. The introduction of cardinality operators can lead to non-deterministic models, for which we use the strategy described in Section 3 to remove this non-determinism.

Input: An XML content model in SF.
Output: A correct XML content model in XNF.

Step 1. Remove redundant choices of the form $R|R$ (rule 1c). Change the order of alternatives if needed (rule 1b).

Step 2. Search for opportunities to introduce cardinality operators, and make sure that this is appropriate in the underlying domain. Find all choices that can be combined, and place these next to each other (rule 1b). If the ϵ alternative is not present, rewrite all choices to R^+ (rule 4b); otherwise, use rule 4a. Afterwards, duplicate alternatives can be removed (rule 1c). Sometimes, parts have to be rewritten before the cardinality operators can be introduced.

Step 3. If no more cardinality operators have to be introduced, bring the expression into XNF by applying factorization and removing ϵ's. The details of this procedure are discussed in Section 3.

Example 6. Consider the model $ab \mid abab \mid abc \mid \epsilon$, which is in SF. We identify three out of four alternatives as instances of $(ab)^*$, i.e., zero or more occurrences of ab. Rewriting the term then proceeds as follows:

$$\begin{aligned}
ab \mid abab \mid abc \mid \epsilon &= \epsilon \mid ab \mid abab \mid abc && \text{(1b)}\\
&\leq (ab)^* \mid (ab)^* \mid (ab)^* \mid abc && \text{(4a)}\\
&= (ab)^* \mid abc && \text{(1c)}\\
&= \epsilon \mid ab(ab)^* \mid abc && \text{(2b)}\\
&= \epsilon \mid ab((ab)^* \mid c) && \text{(3a)}\\
&= (ab((ab)^* \mid c))? && \text{(2a)}
\end{aligned}$$

The resulting model is in XNF. The step in which we give up precision and introduce $(ab)^*$ is made explicit in the derivation, and this is where domain knowledge is required. We could have decided to also rewrite the subexpression abc into $(ab)^*c$, which would lead to the more concise (but less precise) content model $(ab)^*c?$.

5.2 Using the strategy

The strategy for correct models should be used if the number of choices in the SF is large, and a precise content model would be too verbose. The correctness of a final content model can be tested using a validating parser: this parser checks all sequences of child elements in the instance XML-document against the content model, and it will complain if the model is incorrect.

Our strategies are also useful if we start with an informal description of a content model, instead of a starting form. For example, in a chess game, white and black alternate moves, and white has the opening move. These requirements could be translated into the model $(white, black)^+, white?$. The strategy for removing non-determinism (situation 2, case b) suggests to make the model less precise, or to introduce an extra level.

6. EXPERIMENT AND VALIDATION

Five students have participated in a small experiment consisting of a pretest, an online lecture of two hours, a posttest, and an interview. These students have followed the regular bachelor course on XML. The course introduces schema languages, but does not explain any method for modeling content models. All students, except for one, have some basic knowledge about REs or propositional logic.

During the lecture, the relation between content models and REs, the syntax of REs, the language generated by an RE, the rules for rewriting REs, non-determinism and how to remove this, and strategies for modeling precise and correct models were discussed. Students were asked to practice with some exercises. After the lecture, the students had access to the lecture sheets (including examples and exercises).

The pre and posttest consisted of nine questions about modeling precise and correct content models, and removing non-determinism by rewriting or by introducing extra levels in the XML-tree. In the posttest, four questions were repeated from the pretest. The pre and posttest were marked after the students were interviewed: answers were either correct (1 point) or wrong (no score).

6.1 Results

All students scored one or two points higher on the posttest with respect to the pretest; the mean score increased from 4.6 to 6.0, where the maximum score was nine points. Furthermore, we observed a shift in the kind of mistakes. In the pretest, students often produced models that are too liberal for the XML instance document, or models that are not deterministic. Typically, only a final model was given. In the posttest, intermediate steps were given by the students, although not always successfully. Typical mistakes

were the incorrect application of the distribution rule and the empty content missing in the starting form. In addition, we observed an over-carefulness in introducing cardinality operators.

In the interview, the students were asked to what extent the strategies helped in finding precise and correct models. The students reported that the method was particularly useful to get started with complex models. In the pretest, most used a trial-and-error approach. The students also stated that the approach provided a better understanding of precise versus correct models. As a consequence, they think more carefully about introducing cardinality operators.

The participants did not find the method difficult to learn, but they indicated that more practice is needed for applying the rewrite rules and strategies without errors. The students also agreed with the claim that formal methods are lacking in computer science education.

6.2 Discussion

We are careful not to draw strong conclusions based on the tests and the interview. For this, an experiment on a larger scale is needed. We also acknowledge that the students were encouraged to practice with modeling XML content between the pre and posttest, and to study the new material, which also contributes to the improved scores for the posttest. Nevertheless, the students generally welcome the use of formal methods for a practical purpose, and our approach has some clear advantages from an educational point of view.

The first advantage is that it stimulates students to write down a stepwise derivation, and not just a final answer. Once students become more familiar with rewriting models, some trivial steps can be safely skipped. A stepwise approach helps in decomposing a complex task, which is particularly helpful to get started. We observed that many errors during rewriting were not noticed, partly because the students are not accustomed to check their answer. Such a sanity check deserves more attention in teaching the method.

A second advantage is that students are much more aware of the strictness of a content model, and the consequences of introducing cardinality operators. This aspect of schema languages is often overlooked in teaching material on XML.

7. RELATED WORK

Systematic approaches to problem solving play an important role in education. These approaches are often based on three components: knowledge about the domain, means to reason with that knowledge, and a strategy or procedure to guide that reasoning [6, 11]. Our approach is based on making the rewrite rules, and the procedure for using these rules, explicit.

In computer science education, the incorporation of formal methods is strongly suggested by scientific societies such as ACM/IEEE, and many influential scientists [10]. Students employing formal methods during analysis and specification produce more correct, concise, and less complex models [12]. In many curricula, however, formal methods are treated solely as a separate subject to study [9]. Wing et al. [14] advise to weave the use of formal methods into existing courses, making it an additional problem solving technique. We think that our approach is a good example of this advise.

There is an extensive literature about the algorithmic inference of XML content models, and about dealing with nondeterminism [3]. These algorithms often involve the construction of finite-state automata, which makes them more difficult to carry out by hand. We are not aware not other approaches that aim at manually deriving models, at the level of an undergraduate course.

8. CONCLUSIONS AND FUTURE WORK

We have shown that rewrite rules and strategies for regular expressions help students in understanding XML content models, and to guide in the stepwise construction of such a model. The approach makes a sharp distinction between precise and correct models. The first results from using the approach in practice are promising: students appreciate the use of formal methods for solving practical problems. More importantly, they produce better XML content models.

We will proceed our research in the following directions. We plan to further incorporate the method in our course material, and to use it for a larger group of students. Currently, we are also working on methods for the design of algorithms that process XML trees. We are interested in applying our approach to other computer science topics, and to embed the method in a didactic framework.

Acknowledgements. The authors wish to thank Marko van Eekelen, Johan Jeuring, and Lex Bijlsma for their helpful comments on an earlier draft. We are grateful to the students that participated in the experiment.

9. REFERENCES

[1] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge university press, 1999.

[2] G.J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. In *International Conference on World Wide Web*, pages 825–834. ACM, 2008.

[3] G.J. Bex, W. Martens, W. Gelade, and F. Neven. Simplifying XML schema: Effortless handling of nondeterministic regular expressions. In *International Conference on Management of Data*, pages 731–744. ACM, 2009.

[4] A. Bruggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120:87–98, 1993.

[5] A. Bruggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 140:229–253, 1998.

[6] A. Bundy. *The Computer Modelling of Mathematical Reasoning.* Academic Press, 1983.

[7] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[8] J. Jeuring and S.D. Swierstra. *Grammars and Parsing.* Universiteit Utrecht, 2001. Lecture notes.

[9] L. Lamport. The future of computing: logic or biology, 2003. Text of a talk given at Christian Albrechts University, Kiel.

[10] B. Meyer. *Touch of class: Learning to program well with objects and contracts.* Springer, 2009.

[11] J.G. van Merriënboer and P. Kirschner. *Ten steps to complex learning*. Routledge, 2007.

[12] A.E.K. Sobel and M.R. Clarkson. Formal methods application: An empirical tale of software development. *IEEE Transactions on Software Engineering*, 28:308–320, 2002.

[13] D.A. Watt and D.F. Brown. *Programming language processors in Java*. Prentice Hall, 2000.

[14] J.M. Wing and J.M. Wing. Weaving formal methods into the undergraduate computer science curriculum. In *Proceedings of AMAST*, pages 2–9, 2000.