Turning DTDs into specialized tree-automata-based schemata to match a collection of marked-up documents

Rafael C. Carrasco
Dept. de Lenguajes y Sistemas Informáticos
University of Alicante
E-03071 Alicante, Spain
carrasco@dlsi.ua.es

Alejandro Bia
Miguel de Cervantes Digital Library / Dept.L.S.I.
University of Alicante
E-03071 Alicante, Spain
alex.bia@ua.es

Mikel L. Forcada
Dept.L.S.I. / interNOSTRUM
University of Alicante
E-03071 Alicante, Spain
mlf@internostrum.com

ABSTRACT

Regular tree automata (RTA) or, equivalently, forest regular grammars (FRG) have been recently proposed for use as XML (extended markup language) schemata. They are more powerful than usual XML DTDs (document-type definitions), make the implementation, optimization and pruning of XML queries easier and allow for the implementation of context-sensitive content models. We describe a method for the automatic generation of a specialized RTA-based schema from a source DTD and a sample of marked-up documents showing context-sensitive behaviour in content models. It creates the smallest RTA-based schema with which all the XML documents in the sample comply and which does not accept any documents not valid according to the original DTD. In this way, new files can be created, parsed, and queried using the specialized schema but still be compliant with the original DTD. The tool is currently being tested at the Miguel de Cervantes digital library at the University of Alacant (`http://cervantesvirtual.com`).

Keywords

automatic learning, feature extraction, grammatical inference, document analysis, document markup, digital libraries, tree automata, document type definitions

1. INTRODUCTION

An extended markup language (XML) document-type definition (DTD) specifies the elements that are allowed in a document of this type. Document types are defined by extended context-free grammars in which the right-hand sides of productions are unambiguous regular expressions [8]. To address some of the problems posed by XML DTDs, regular tree automata (RTA), or, equivalently, forest regular grammars (FRG), have been recently proposed for use as XML schemata[13, 19]; RTA-based schemata are more powerful than usual XML DTD. For example, XML queries [13] are more easily implemented, optimized and pruned in terms of RTA-based schemata. In particular, RTA-based schemata allow the implementation of context-sensitive content models [19], which allow an element to have different behaviors depending on its placement in the document structure, i.e. depending on its ancestors in the document hierarchy. For instance, it is common sense to expect paragraphs to be allowed to have footnotes in general except when they are included within certain particular structures as could be an abstract or a metadata section, such as the `teiHeader` of the TEI (Text Encoding Initiative) encoding scheme [21, 18, 10], without having to label each kind of element with a different name in the text itself. Regular tree automata and languages also provide a powerful formalism to address document and schema transformations [16].

In the ensuing discussion, we will use the following DTD (inspired in [19]) as an example:

```
<!ELEMENT DOC (HEAD,BODY)>
<!ELEMENT HEAD (TITLE,ABSTRACT)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT ABSTRACT (P)+>
<!ELEMENT BODY (P)+>
<!ELEMENT P (#PCDATA,(FOOTNOTE,#PCDATA)*)>
<!ELEMENT FOOTNOTE (#PCDATA)>
```

1.1 Regular tree automata

A *regular tree automaton* [13] is a 5-tuple $M = (\Sigma, D, Q, \delta, F)$ where Σ is a finite set of element types, D is a finite set of data types, Q is a finite set of states, δ is a function $\delta : \Sigma \times E \rightarrow Q$ where E is either an element of D or a regular expression over $Q \cup D$ (this is a slight variation from the definition in [13]), and $F \subset Q$ is the set of final states. The automaton walks the document tree bottom-up and computes, using function $\delta()$ a state at each node from the label at the node itself and the states of its children and accepts a document tree only if the state eventually computed for the root node is in F. The DTD above may be turned into a regular tree automaton in which $\Sigma = \{\texttt{DOC}, \texttt{HEAD}, \texttt{TITLE}, \texttt{ABSTRACT}, \texttt{BODY}, \texttt{P}, \texttt{FOOTNOTE}\}$, $D = \{\#\texttt{PCDATA}\}$, $Q = \{q_{\texttt{DOC}}, q_{\texttt{HEAD}}, q_{\texttt{TITLE}}, q_{\texttt{ABSTRACT}}, q_{\texttt{BODY}}, q_{\texttt{P}}, q_{\texttt{FOOTNOTE}}\}$, $F = \{q_{\texttt{DOC}}\}$,

Miguel de Cervantes DL, R&D dept.
PRELIMINARY DRAFT DOCUMENT (for Markup Languages journal)

and

$$\begin{aligned}
\delta(\texttt{DOC}, q_{\texttt{HEAD}}\ q_{\texttt{BODY}}) &= q_{\texttt{DOC}} \\
\delta(\texttt{HEAD}, q_{\texttt{TITLE}}\ q_{\texttt{ABSTRACT}}) &= q_{\texttt{HEAD}} \\
\delta(\texttt{TITLE}, \#\texttt{PCDATA}) &= q_{\texttt{TITLE}} \\
\delta(\texttt{ABSTRACT}, q_{\texttt{P}}^{+}) &= q_{\texttt{ABSTRACT}} \\
\delta(\texttt{BODY}, q_{\texttt{P}}^{+}) &= q_{\texttt{BODY}} \\
\delta(\texttt{P}, \#\texttt{PCDATA}(q_{\texttt{FOOTNOTE}}\ \#\texttt{PCDATA})^{*}) &= q_{\texttt{P}} \\
\delta(\texttt{FOOTNOTE}, \#\texttt{PCDATA}) &= q_{\texttt{FOOTNOTE}}
\end{aligned} \tag{1}$$

where it can be seen that there is a one-to-one correspondence between states in Q and elements in Σ; this is not the general case, but is instead due to the fact that the RTA has been derived from a DTD.

1.2 Forest regular grammars

A forest regular grammar (FRG, [19]) $G = (Q, \Sigma, D, P, S)$ has a finite set of variables Q, a finite set of labels Σ, a finite set of data types D, a set of productions P of the form $V \rightarrow$ <Σ>α</Σ>, where α is any regular expression over $Q \cup D$, and a start symbol $S \in Q$. The RTA in eq. (1) may be written as a FRG having variables $Q = \{q_{\texttt{DOC}}, q_{\texttt{HEAD}}, q_{\texttt{TITLE}}, q_{\texttt{ABSTRACT}}, q_{\texttt{BODY}}, q_{\texttt{P}}, q_{\texttt{FOOTNOTE}}\}$, document labels $\Sigma = \{\texttt{DOC}, \texttt{HEAD}, \texttt{TITLE}, \texttt{ABSTRACT}, \texttt{BODY}, \texttt{P}, \texttt{FOOTNOTE}\}$, data types $D = \{\#\texttt{PCDATA}\}$, $S = \{\texttt{DOC}\}$ and the following set of productions P: follows:

$$\begin{aligned}
q_{\texttt{DOC}} &\rightarrow \texttt{<DOC>}\ q_{\texttt{HEAD}}\ q_{\texttt{BODY}}\ \texttt{</DOC>} \\
q_{\texttt{HEAD}} &\rightarrow \texttt{<HEAD>}\ q_{\texttt{TITLE}}\ q_{\texttt{ABSTRACT}}\ \texttt{</HEAD>} \\
q_{\texttt{TITLE}} &\rightarrow \texttt{<TITLE>}\ \#\texttt{PCDATA}\ \texttt{</TITLE>} \\
q_{\texttt{ABSTRACT}} &\rightarrow \texttt{<ABSTRACT>}\ q_{\texttt{P}}^{+}\ \texttt{</ABSTRACT>} \\
q_{\texttt{BODY}} &\rightarrow \texttt{<BODY>}\ q_{\texttt{P}}^{+}\ \texttt{</BODY>} \\
q_{\texttt{P}} &\rightarrow \texttt{<P>}\ \#\texttt{PCDATA}\ (q_{\texttt{FOOTNOTE}}\ \#\texttt{PCDATA})^{*}\ \texttt{</P>} \\
q_{\texttt{FOOTNOTE}} &\rightarrow \texttt{<FOOTNOTE>}\ \#\texttt{PCDATA})\ \texttt{</FOOTNOTE>}
\end{aligned} \tag{2}$$

As in the case of the RTA in eq. (1), the fact that the FRG is derived from a DTD results in a set of states Q which is essentially the same as the set of labels Σ.

1.3 Beyond DTDs

But regular-tree automata and forest regular grammars are more powerful than what these examples show because the elements in Q need not be the same as the labels in Σ. In particular, one could choose elements of Q to be partitions or specializations of the labels in Σ. Let us consider the example considered before, in which paragraphs may have footnotes in general but do not have footnotes when they are inside an abstract. It is not possible to write an XML DTD for this case, but it is quite straightforward to write a RTA or a FRG for it; only the FRG will be shown:

$$\begin{aligned}
q_{\texttt{DOC}} &\rightarrow \texttt{<DOC>}\ q_{\texttt{HEAD}}\ q_{\texttt{BODY}}\ \texttt{</DOC>} \\
q_{\texttt{HEAD}} &\rightarrow \texttt{<HEAD>}\ q_{\texttt{TITLE}}\ q_{\texttt{ABSTRACT}}\ \texttt{</HEAD>} \\
q_{\texttt{TITLE}} &\rightarrow \texttt{<TITLE>}\ \#\texttt{PCDATA}\ \texttt{</TITLE>} \\
q_{\texttt{ABSTRACT}} &\rightarrow \texttt{<ABSTRACT>}\ q_{\texttt{P_NOFN}}^{+}\ \texttt{</ABSTRACT>} \\
q_{\texttt{BODY}} &\rightarrow \texttt{<BODY>}\ q_{\texttt{P}}^{+}\ \texttt{</BODY>} \\
q_{\texttt{P}} &\rightarrow \texttt{<P>}\ \#\texttt{PCDATA}\ (q_{\texttt{FOOTNOTE}}\ \#\texttt{PCDATA})^{*}\ \texttt{</P>} \\
q_{\texttt{P_NOFN}} &\rightarrow \texttt{<P>}\ \#\texttt{PCDATA}\ \texttt{</P>} \\
q_{\texttt{FOOTNOTE}} &\rightarrow \texttt{<FOOTNOTE>}\ \#\texttt{PCDATA})\ \texttt{</FOOTNOTE>}
\end{aligned} \tag{3}$$

In this FRG, state $q_{\texttt{P\_NOFN}}$ is the special case (no footnotes inside an abstract), whereas $q_{\texttt{P}}$ is the general case.

1.4 Previous work

Previous work has addressed the task of identifying a DTD from examples of XML documents.[1] A common difficulty in this approach is the need to find a correct degree of generalization. Some practical tools as FRED [20] let the users customize their preferred degree of generalization. Ahonen [2, 4] builds a (k, h)-testable model for the element contents and needs non-trivial further generalization in order to disambiguate the model [3]. Young-Lai and Tompa [23] rely on a stochastic approach to control overgeneralization, based in turn on the algorithm by Carrasco and Oncina [12]. Presumably, the stochastic approach needs large collections of hand-tagged documents. Pizza-Chef [9] is a tool to generate TEI-compliant DTDs customized to satisfy specific markup needs by means of an HTML form with optional pizza-restaurant-like choices; although it is an excellent tool, it does not automatically learn a DTD from examples, and is not a general purpose tool, being limited to produce DTDs derived only from the TEI encoding scheme. More recently, Bia, Carrasco and Forcada [6] have proposed a method to automatically select from a source DTD only those features that are used by a set of valid sample documents and eliminate the rest of them, obtaining a narrow-scope DTD which defines a subset of the original markup scheme. This pruned DTD can be used to build new documents of the same markup subclass. The simplified DTD can be updated immediately in the event that new features are added to (or even eliminated from) the sample set of XML files. This process can be repeated as often as needed to generate an updated DTD. In particular, Bia et al.'s [6] technique may be used to build a one-document DTD, i.e. the minimal markup schema derived from the general DTD with which a given XML document complies.

1.5 Objective

This paper describes a method for the automatic generation of a specialized RTA-based schema from a source DTD and a sample of marked-up documents showing context-sensitive behaviour in content-models. The purpose is to create the smallest RTA-based schema with which the whole sample of XML documents complies and which in turn does not accept any documents not valid according to the original DTD. In this way, new files can be created, parsed, and processed for queries [13] using the specialized schema but still being compliant with the original, DTD; for example, some particular context-sensitivities observed a given set of digital library texts, encoded following a general XML DTD such as `teixlite.dtd` may be better described and processed with the new RTA-based schema. The tool is currently being tested at the Miguel de Cervantes digital library[2]. This work is part of a larger project in the field of text markup and derived applications [7, 5, 6].

The method proposed in this paper aims at obtaining a close description of a set of XML documents, but at the price of having to use a more powerful formulation, that of RTA-based schemata; it goes one step further from [5] by allowing elements to have content models that depend on the particular context where the element appears.

A further application of this technique is to generate statistics that may help DTD designers improve their markup schemes. Information about the frequency of use of certain

[1] DTD inference from XML queries has been addressed by [17] and [15].

[2] `http://cervantesvirtual.com/`

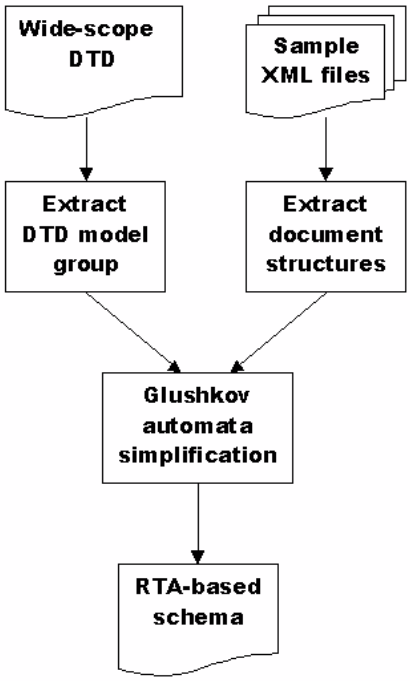

Figure 1: *Architecture of the tool to build RTA-based schemata*

elements within others helps us to detect unusual structures that could reflect markup mistakes or DTD features that allow for unwanted generalization.

The advantages of following a de-facto standard, or at least a commonly used markup scheme amongst digital libraries (as the TEI scheme is), like text interchangeability or inter-project cooperation —just to mention a few—, the goal of using XML from the very start of the project, as well as the need to spare the cost of developing our own DTD are some of the reasons why the `teixlite.dtd` has been chosen at the Miguel de Cervantes Digital Library. But even such a simple XML DTD, complete enough to structurally markup prose, verse and drama, is too complex for markup beginners, contains innecessary elements, and allows the insertion of tags in contexts where they are not desirable. On the other hand, it contains only element definitions that are independent of the context, while context-dependencies are common in texts. In the future, as we plan to implement advanced (Boolean) queries, we need to go beyond the DTD formulation [13]. The method proposed in this paper allows us to convert the `teixlite.dtd` into a RTA-based schema which still allows only documents that are TEI-compliant and thus benefit from the advantages of sharing a common DTD with other international digitization projects.

We started by defining what kinds of modifications will be allowed in order to make markup simpler to use but keeping TEI compatibility (except for minor exceptions). In particular, we allowed for the following changes:

- To specify a set of normalized values for some attributes in order to enforce their use instead of free data entry.
- To add new attributes only in a few necessary cases (this is the only exception that may keep our files from being TEI compliant, but they can be easily removed anytime we want full TEI compatibility).
- To impose restrictions in element inclusion rules in order to eliminate the possibility of including certain elements at certain levels of the markup.
- To make some optional elements or attributes mandatory, following our specific markup norms.
- To eliminate optional elements we will not use to simplify the markup task and to avoid possible errors.

It may be easily admitted that simplifying a DTD and turning it into a specialized RTA-based schema by hand is tedious and error-prone. Using our method, the task may however be performed automatically by processing a set of sample documents representative of all kinds of documents we need to markup together for a given application.

2. THE PROCESS

A diagram of the process —a generalization of the one described in [5, 6]— is shown in figure 1. The key notion is that of *element-in-context* (EiC), that is, a specialized version of an element for each possible element enclosing it (for instance, a paragraph (`P`) inside an `ABSTRACT` may be described by the EiC `P-IN-ABSTRACT`, which is however still marked `<P>...</P>` in the text. As the diagram shows, the initial DTD is processed to extract the structure of the markup models and a Glushkov automaton [11] is built for each one (that is, for each regular expression).

The XML sample files are then preprocessed to extract the EiCs; that is, to label the contexts (content models) in which these elements are used, and to establish their nesting patterns.[3] We keep track of the EiCs used in the sample files and mark the visited states of the Glushkov automaton corresponding to each content model for each possible EiC, to simplify the corresponding regular expressions. Finally, we eliminate unused EiCs, merge EiCs having identical content models into single EiCs having more general contexts (which will constitute the states of the RTA or the variables of the FRG), and update the names of EiCs in the right parts of element-in-context definitions to account for these mergings. It has to be stressed that the right parts of all EiC are all simplified (pruned) versions of the regular expressions defining the content of the corresponding context-free element in the original DTD, with simplifications being different in each context.

For the implementation of the the RTA-based-schemata building toolkit we need both an XML parser and a DTD parser. We assume that both the XML sample files and the source DTD are well-formed and already valid, so there is no need to build validating parsers. In particular, regular expressions are parsed against the extended Backus-Naur form (EBNF) grammar described in the following section, although indeed, XML forces stricter parentization patterns.

2.1 An example

Consider a set of XML documents and the following DTD defining them:

[3]The concept of EiC is closely related to that of k-testable tree languages as introduced by Knuutila [14] and Verdú et al. [22].

```
<!ELEMENT TEXT (ABSTRACT,BODY)>
<!ELEMENT ABSTRACT (P)+>
<!ELEMENT BODY (P)+>
<!ELEMENT P (#PCDATA,(FOOTNOTE,#PCDATA)*)>
<!ELEMENT FOOTNOTE (P)+>
```

The DTD is first converted into a rather redundant RTA-based schema (shown here as a FRG) which is equivalent to the DTD but uses EiCs.

$$\begin{array}{rcl}
q_{\texttt{TEXT}} & \rightarrow & \texttt{<TEXT>}\ q_{\texttt{ABSTRACT-IN-TEXT}}\ q_{\texttt{BODY-IN-TEXT}}\ \texttt{</TEXT>} \\
q_{\texttt{ABSTRACT-IN-TEXT}} & \rightarrow & \texttt{<ABSTRACT>}\ q^{+}_{\texttt{P-IN-ABSTRACT}}\ \texttt{</ABSTRACT>} \\
q_{\texttt{BODY-IN-TEXT}} & \rightarrow & \texttt{<BODY>}\ q^{+}_{\texttt{P-IN-BODY}}\ \texttt{</BODY>} \\
q_{\texttt{P-IN-ABSTRACT}} & \rightarrow & \texttt{<P>}\ \#\texttt{PCDATA}\ (q_{\texttt{FOOTNOTE-IN-P}}\ \#\texttt{PCDATA})^{*}\ \texttt{</P>} \\
q_{\texttt{P-IN-BODY}} & \rightarrow & \texttt{<P>}\ \#\texttt{PCDATA}\ (q_{\texttt{FOOTNOTE-IN-P}}\ \#\texttt{PCDATA})^{*}\texttt{</P>} \\
q_{\texttt{P-IN-FOOTNOTE}} & \rightarrow & \texttt{<P>}\ \#\texttt{PCDATA}\ (q_{\texttt{FOOTNOTE-IN-P}}\ \#\texttt{PCDATA})^{*}\texttt{</P>} \\
q_{\texttt{FOOTNOTE-IN-P}} & \rightarrow & \texttt{<FOOTNOTE>}\ q_{\texttt{P-IN-FOOTNOTE}}\ \texttt{</FOOTNOTE>}
\end{array} \tag{4}$$

As may be seen, the only element appearing in more than one context is `P`, which appears as `P-IN-ABSTRACT`, `P-IN-BODY`, and `P-IN-FOOTNOTE`. Then this schema is used to parse the set of XML documents to mark those parts in the regular expressions describing the content models that have been used. Let us assume that it is the case that footnotes never appear in abstracts or in footnotes. In this case, after executing the regular expressions simplifications described in detail in sections 3 and 4, one would obtain a new FRG:

$$\begin{array}{rcl}
q_{\texttt{TEXT}} & \rightarrow & \texttt{<TEXT>}\ q_{\texttt{ABSTRACT-IN-TEXT}}\ q_{\texttt{BODY-IN-TEXT}}\ \texttt{</TEXT>} \\
q_{\texttt{ABSTRACT-IN-TEXT}} & \rightarrow & \texttt{<ABSTRACT>}\ q^{+}_{\texttt{P-IN-ABSTRACT}}\ \texttt{</ABSTRACT>} \\
q_{\texttt{BODY-IN-TEXT}} & \rightarrow & \texttt{<BODY>}\ q^{+}_{\texttt{P-IN-BODY}}\ \texttt{</BODY>} \\
q_{\texttt{P-IN-ABSTRACT}} & \rightarrow & \texttt{<P>}\ \#\texttt{PCDATA}\ \texttt{</P>} \\
q_{\texttt{P-IN-BODY}} & \rightarrow & \texttt{<P>}\ \#\texttt{PCDATA}\ (q_{\texttt{FOOTNOTE-IN-P}}\ \#\texttt{PCDATA})^{*}\texttt{</P>} \\
q_{\texttt{P-IN-FOOTNOTE}} & \rightarrow & \texttt{<P>}\ \#\texttt{PCDATA}\ \texttt{</P>} \\
q_{\texttt{FOOTNOTE-IN-P}} & \rightarrow & \texttt{<FOOTNOTE>}\ q_{\texttt{P-IN-FOOTNOTE}}\ \texttt{</FOOTNOTE>}
\end{array} \tag{5}$$

Which is finally simplified by merging those EiCs having the same content models, eliminating context information from EiC names where it is not necessary, and substituting the resulting EiC names in content models. The resulting 6-rule FRG is:

$$\begin{array}{rcl}
q_{\texttt{TEXT}} & \rightarrow & \texttt{<TEXT>}\ q_{\texttt{ABSTRACT}}\ q_{\texttt{BODY}}\ \texttt{</TEXT>} \\
q_{\texttt{ABSTRACT}} & \rightarrow & \texttt{<ABSTRACT>}\ q^{+}_{\texttt{P-IN-ABSTRACT-FOOTNOTE}}\ \texttt{</ABSTRACT>} \\
q_{\texttt{BODY}} & \rightarrow & \texttt{<BODY>}\ q^{+}_{\texttt{P-IN-BODY}}\ \texttt{</BODY>} \\
q_{\texttt{P-IN-ABSTRACT-FOOTNOTE}} & \rightarrow & \texttt{<P>}\ \#\texttt{PCDATA}\ \texttt{</P>} \\
q_{\texttt{P-IN-BODY}} & \rightarrow & \texttt{<P>}\ \#\texttt{PCDATA}\ (q_{\texttt{FOOTNOTE}}\ \#\texttt{PCDATA})^{*}\ \texttt{</P>} \\
q_{\texttt{FOOTNOTE}} & \rightarrow & \texttt{<FOOTNOTE>}\ q_{\texttt{P-IN-ABSTRACT-FOOTNOTE}}\ \texttt{</FOOTNOTE>}
\end{array} \tag{6}$$

with only two types of paragraph. All documents conforming to this schema comply also with the original DTD, but only those documents which both comply with the DTD and do not have footnotes in abstracts or footnotes will comply with the new schema.

3. THEORETICAL FOUNDATION OF REGULAR EXPRESSION SIMPLIFICATION

The set $\mathrm{reg}(\Sigma)$ of regular expressions over a finite or infinite alphabet $\Sigma = \{a_1, a_2, \ldots\}$ can be defined as the language generated by the context-free grammar (V, T, R, E) with rules

$$\begin{array}{rcl}
E & \rightarrow & T \mid T\text{“|”}E \\
T & \rightarrow & F \mid F\text{“,”}T \\
F & \rightarrow & W \mid W\text{“}*\text{”} \mid W\text{“}+\text{”} \mid W\text{“?”} \\
W & \rightarrow & \text{“(”}E\text{“)”} \\
W & \rightarrow & a_1 \mid a_2 \mid \ldots
\end{array} \tag{7}$$

and terminals $T = \Sigma \cup \{\text{“|”}, \text{“,”}, \text{“}*\text{”}, \text{“}+\text{”}, \text{“?”}, \text{“(”}, \text{“)”}\}$. For every regular expression $r \in \mathrm{reg}(\Sigma)$, we denote with $\mathrm{sym}(r) \subseteq \Sigma$ the subset of symbols used in r.

A *marking* of r is a pair (Φ_r, E_r) with

- $E_r \in \mathrm{reg}(\mathbb{N})$ such that no $n \in \mathbb{N}$ is used in E_r more than once;
- $\Phi_r : \mathbb{N} \rightarrow \Sigma$ is a mapping such that r is the result of replacing every symbol $n \in \mathrm{sym}(E_r)$ in E_r (called *positions*) with $\Phi_r(n)$.

For instance, if $\Sigma = \{a, b\}$ and $r = ((a, b)|a)*$, a marking of r is given by $E_r = ((1, 2)|3)*$ with $\Phi_r(1) = \Phi_r(3) = a$ and $\Phi_r(2) = b$. We can immediately extend Φ_r to work on subexpressions of E_r if we assume that Φ_r is a homomorphism such that $\Phi_r(E_r) = r$.

The XML standard requires the regular expressions describing the possible content of an element (that is, its *content model*) to be unambiguous in the following sense: an element or string in the document is witnessed without lookahead by at most one token in the regular expression; unambiguity is required to avoid the construction of exponentially-large parsers. The method used here to simplify regylra expressions ensures that the content models corresponding to the elements-in-context in a RTA-based schema will have the same property.

More precisely, a regular expression r is *1-unambiguous* if for all $x, y, z \in \mathbb{N}^*$ (i.e., finite strings of naturals) and for all $n, m \in \mathbb{N}$

$$\left.\begin{array}{c} xny \in L(E_r) \\ xmz \in L(E_r) \\ n \neq m \end{array}\right\} \Rightarrow \Phi_r(xny) \neq \Phi_r(xmy) \tag{8}$$

The definition above can be formulated in an alternative fashion as follows.

THEOREM 1. *(Lemma 2.5 in [8]) A regular expression r is 1-unambiguous if and only if the Glushkov automaton of r is deterministic.*

Details on how to build the Glushkov automaton for a given expression r can be found in the appendix and in [11]. The next theorem supports the validity of our simplification process.

THEOREM 2. *Let r be a 1-unambiguous regular expression and $f(E_r)$ denote the result of a homomorphism that replaces some positions in E_r by the empty set symbol $\emptyset$. Then, $\Phi_r(f(E_r))$ is 1-unambiguous.*

Proof: Let $\mu \subset \mathrm{sym}(E_r)$ be the subset of positions n in E_r such that $f(n) \neq \emptyset$. Then, $L(f(E_r)) = L(E_r) \cap L(\mathrm{reg}(\mu))$ and then $L(f(E_r)) \subseteq L(E_r)$. Therefore, we may substitute $L(E_r)$ by $L(f(E_r))$ in definition (8) and the implication remains valid. Then, $\Phi_r(f(E_r))$ is 1-unambiguous.

4. REGULAR EXPRESSION PRUNING

The process by means of which the regular expression describing the content of each EiC is simplified is based on a bottom-up parse of the original regular expression. A syntax-directed definition [1] describing this process is shown in Appendix B . The process replaces any unwitnessed position in the expression E_r by the regular expression corresponding to the empty set ($\emptyset$); then the expression is projected into the reg(Σ) space; finally, the resulting regular expression is rearranged to avoid using symbols not in Σ.

The following simplification rules, used in the last step, preserve unambiguity as the resulting expression after each replacement defines exactly the same language:

$$\begin{array}{rclcl} \emptyset, E & = & E, \emptyset & = & \emptyset+ = \emptyset \\ \emptyset|E & = & E|\emptyset & = & E \\ \emptyset* & = & \emptyset? & = & \lambda \\ \lambda, E & = & E, \lambda & = & E \\ \lambda|E & = & E|\lambda & = & \begin{cases} E & \text{if empty}(E) \\ E? & \text{otherwise} \end{cases} \\ \lambda* & = & \lambda+ & = & \lambda? = \lambda \end{array} \tag{9}$$

Here, λ is a special symbol denoting the empty string, not allowed in a valid regular expression, and empty() is a Boolean function determining whether the regular expression accepts the empty string or not (the way to compute it efficiently can be found in the appendix and in [11]). The resulting expression is a simplified one containing neither $\emptyset$ nor λ symbols, except when the result is exactly $\emptyset$, in which case the element will be removed or an empty content model will be substituted.

5. CONCLUSIONS AND FUTURE WORK

We have developed at the Miguel de Cervantes Digital Library a method which has been used to automatically generate RTA-based XML schemata from general DTD. On this first stage, we have addressed the simplification of element type descriptions based on sample files and the use of context-sensitive content models for elements, called elements in context. On a second stage, we plan to add automatic elimination or addition of attributes. We also plan to collect statistics to detect unusual patterns that may reflect markup mistakes.

6. REFERENCES

[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, Massachusetts, USA, 1986.

[2] H. Ahonen. Automatic Generation of SGML Content Models. *Electronic Publishing Origination, Dissemination, and Design*, 8(2/3):195–206, June/Sept. 1995.

[3] H. Ahonen. Disambiguation of SGML content models. *Lecture Notes in Computer Science*, 1293:27, 1997.

[4] H. Ahonen, H. Mannila, and E. Nikunen. Generating grammars for SGML tagged texts lacking DTD. *Mathematical and Computer Modelling*, 26(1):1–13, 1997.

[5] A. Bia and R. C. Carrasco. Automatic DTD simplification by examples. In *ACH/ALLC 2001. The Association for Computers and the Humanities, The Association for Literary and Linguistic Computing, The 2001 Joint International Conference*, pages 7–9, New York University, New York City, June 2001.

[6] A. Bia, R. C. Carrasco, and M. L. Forcada. Identifying a reduced DTD from marked up documents. In J. S. Sánchez and F. Pla, editors, *Pattern Recognition and Image Analysis (Proceedings of the IX Spanish Symposium on Pattern Recognition and Image Analysis – SNRFAI'2001 conference)*, volume II of *Treballs d'informática i tecnologia, num.7*, pages 385–390, Castellón de la Plana, Spain, May 2001. Publicacions de la Universitat Jaume I, 2001.

[7] A. Bia and A. Pedreño. The Miguel de Cervantes Digital Library: the Hispanic voice on the Web. *LLC (Literary and Linguistic Computing) journal, Oxford University Press*, 16(2):161–177, 2001. Presented at ALLC/ACH 2000, The Joint International Conference of the Association for Literary and Linguistic Computing and the Association for Computers and the Humanities, 21/25 July 2000, University of Glasgow.

[8] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1 May 1998.

[9] L. Burnard. The Pizza Chef: a TEI Tag Set Selector. `http://www.hcu.ox.ac.uk/TEI/pizza.html`, September 1997. (Original version 13 September 1997, updated July 1998; Version 2 released 8 Oct 1999).

[10] L. Burnard and C. M. Sperberg-McQueen. Tei lite: An introduction to text encoding for interchange (document no: Tei u 5). http://www.uic.edu/orgs/tei/intros/teiu5.html, 6 1995.

[11] P. Caron and D. Ziadi. Characterization of Glushkov automata. *TCS: Theoretical Computer Science*, 233:75–90, 2000.

[12] R. C. Carrasco and J. Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. *RAIRO (Theoretical Informatics and Applications)*, 33(1):1–20, 1999.

[13] B. Chidlovskii. Using Regular Tree Automata as XML Schemas. In J. Hoppenbrouwers, T. de Souza Lima, M. Papazoglou, and A. Sheth, editors, *Proc. IEEE Advances on Digital Libraries Conference 2000*, pages 89–98, 2000.

[14] T. Knuutila. Inference of k-testable tree languages. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition (Proc. Intl. Workshop on Structural and Syntactic Pattern Recognition, Bern, Switzerland)*. World Scientific, August 1993.

[15] B. Ludaescher, Y. Papakonstantinou, P. Velikhov, and V. Vianu. View Definition and DTD Inference for XML. In *Proc. Post-ICDT Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, 1999.

[16] M. Murata. Transformation of documents and schemas by patterns and contextual conditions. In D. W. Charles K. Nicholas, editor, *Proceedings of the Third International Workshop on Principles of Document Processing (PODP'96), Palo Alto, California, USA*, volume 1293 of *Lecture Notes in Computer Science*, pages 153–169. Springer-Verlag, Heidelberg, September 23 1996.

[17] Y. Papakonstantinou and V. Vianu. DTD Inference for Views of XML Data. In *Principles Of Database Systems (PODS) 2000*, 2000.

[18] W. Plotkin and C. Sperberg-McQueen. Text Encoding Initiative, homepage (tei@uic.edu). http://www.uic.edu/orgs/tei/index.html.

[19] P. Prescod. Formalizing XML and SGML Instances with Forest Automata Theory. Technical report, University of Waterloo, Department of Computer Science, Waterloo, Ontario, May 1998. Draft Technical Paper.

[20] K. E. Shafer. Creating DTDs via the GB-Engine and Fred. Technical report, OCLC Online Computer Library Center, Inc., 6565 Frantz Road, Dublin, Ohio 43017-3395, 1995.

[21] C. M. Sperberg-McQueen and L. Burnard, editors. *Guidelines for Electronic Text Encoding and Interchange (Text Encoding Initiative P3), Revised Reprint, Oxford, May 1999*. TEI P3 Text Encoding Initiative, Chicago - Oxford, May 1994.

[22] J. Verdú-Mas, J. Calera-Rubio, and R. Carrasco. A comparison of PCFG models. In C. Cardie, W. Daelemans, C. Nédellec, and E. Tjong-Kim-Sang, editors, *Proceedings of CoNLL-2000 and LLL-2000*, pages 123–125, New Brunswick, NJ (USA), 2000. Association for Computational Linguistics.

[23] M. Young-Lai and F. W. M. Tompa. Stochastic Grammatical Inference of Text Database Structure. *Machine Learning*, 40(2):1, 2000.

APPENDIX

A. BUILDING THE GLUSHKOV AUTOMATA OF AN EXPRESSION

Given a regular expression r we describe the procedure [11] to build a Glushkov automaton. In the following $n \in \mathbb{N}$ is any position according to a marking E_r of r; $F, G \in \mathrm{reg}(\mathbb{N})$ denote subexpressions of E_r.

The Boolean function $\mathrm{empty} : \mathrm{reg}(\mathbb{N}) \to \{\mathrm{TRUE}, \mathrm{FALSE}\}$ is true if the subexpression contains the empty string and can be computed as

$$\begin{array}{rcl} \mathrm{empty}(n) & = & \mathrm{FALSE} \\ \mathrm{empty}(F|G) & = & \mathrm{empty}(F) \vee \mathrm{empty}(G) \\ \mathrm{empty}(F,G) & = & \mathrm{empty}(F) \wedge \mathrm{empty}(G) \\ \mathrm{empty}(F*) & = & \mathrm{TRUE} \\ \mathrm{empty}(F+) & = & \mathrm{empty}(F) \\ \mathrm{empty}(F?) & = & \mathrm{TRUE} \end{array} \tag{10}$$

The subset $\mathrm{first} : \mathrm{reg}(\mathbb{N}) \to 2^{\mathbb{N}}$ gives the positions that can be found as the first symbol in a string and is given by

$$\begin{array}{rcll} \mathrm{first}(n) & = & \{n\} & \\ \mathrm{first}(F|G) & = & \mathrm{first}(F) \cup \mathrm{first}(G) & \\ \mathrm{first}(F,G) & = & \begin{cases} \mathrm{first}(F) \cup \mathrm{first}(G) & \text{if } \mathrm{empty}(F) \\ \mathrm{first}(F) & \text{otherwise} \end{cases} & \\ \mathrm{first}(F*) & = & \mathrm{first}(F) & \\ \mathrm{first}(F+) & = & \mathrm{first}(F) & \\ \mathrm{first}(F?) & = & \mathrm{first}(F) & \end{array} \tag{11}$$

The subset $\mathrm{last} : \mathrm{reg}(\mathbb{N}) \to 2^{\mathbb{N}}$ gives the positions that can be found as the last symbol in a string and is given by

$$\begin{array}{rcll} \mathrm{last}(n) & = & \{n\} & \\ \mathrm{last}(F|G) & = & \mathrm{last}(F) \cup \mathrm{last}(G) & \\ \mathrm{last}(F,G) & = & \begin{cases} \mathrm{last}(F) \cup \mathrm{last}(G) & \text{if } \mathrm{empty}(G) \\ \mathrm{last}(G) & \text{otherwise} \end{cases} & \\ \mathrm{last}(F*) & = & \mathrm{last}(F) & \\ \mathrm{last}(F+) & = & \mathrm{last}(F) & \\ \mathrm{last}(F?) & = & \mathrm{last}(F) & \end{array} \tag{12}$$

Finally, the subset $\mathrm{follow} : \mathrm{reg}(\mathbb{N}) \to 2^{\mathbb{N}\times\mathbb{N}}$ gives the pairs of positions that can be found consecutive in a string and is given by

$$\begin{array}{rcl} \mathrm{follow}(n) & = & \emptyset \\ \mathrm{follow}(F|G) & = & \mathrm{follow}(F) \cup \mathrm{follow}(G) \\ \mathrm{follow}(F,G) & = & \mathrm{follow}(F) \cup \mathrm{follow}(G) \cup \mathrm{last}(F) \times \mathrm{first}(G) \\ \mathrm{follow}(F*) & = & \mathrm{follow}(F) \cup \mathrm{last}(F) \times \mathrm{first}(F) \\ \mathrm{follow}(F+) & = & \mathrm{follow}(F) \cup \mathrm{last}(F) \times \mathrm{first}(F) \\ \mathrm{follow}(F?) & = & \mathrm{follow}(F) \end{array} \tag{13}$$

With the above elements, the Glushkov automaton [8] $(Q, \mathbb{N}, \delta, I, F)$ is given by

- $Q = \{I\} \cup \mathrm{sym}(E_r)$
- $\delta(I, a) = \{n \in \mathrm{first}(E_r) : \Phi_r(n) = a\}$
- $\delta(n, a) = \{m \in Q : (n, m) \in \mathrm{follow}(E_r) \wedge \Phi_r(m) = a\}$
- $F = \begin{cases} \{I\} \cup \mathrm{last}(E_r) & \text{if } \mathrm{empty}(E_r) \\ \mathrm{last}(E_r) & \text{otherwise} \end{cases}$

where I is any position not in $\mathrm{sym}(E_r)$.

B. SYNTAX-DIRECTED DEFINITION

The process of simplification is described here as a syntax-directed definition [1] which associates translation actions to each rule in grammar 7, actions which compute attributes at each node of the parse tree of the regular expressions from attributes at child nodes. The simplified regular expression is stored in attribute $.s$, whereas attributes $.n$ and $.e$ are auxiliary Boolean attributes which store, respectively, whether the subexpression is simple and whether it accepts the empty string. Function HasWitnesses() returns TRUE if the symbol in the regular expression has been visited during the parsing of a document in the sample.

PRODUCTION	TRANSLATION
$E_0 \rightarrow T \mid E_1$	$E_0.n := T.n$ **and** $E_1.n$; $E_0.e := T.e$ **or** $E_1.e$; **if** $T.w = \emptyset$ **then** $E_0.w := E_1.w$ **else if** $T.w = \lambda$ **then if** $E_1.w = \emptyset$ **or** $T.w = \lambda$ **or** $E_1.e$ **then** $E_0.w := E_1.w$ **else** $E_0.w := E_1.w \|\|$"?" **endif** **else if** $E_1.w = \lambda$ **then if** $T.e$ **then** $E_0.w := T.w$ **else** $E_0.w := T.w \|\|$"?" **endif** **else if** $E_1.w \neq \emptyset$ **then begin** $E_0.w := T.w \|\|$"|"$\|\|E_1.w$; $E_1.n :=$ FALSE **end** **endif** **endif** **endif** **endif**
$E \rightarrow T$	$E.w := T.w$; $E.n := T.n$; $E.e := T.e$
$T_0 \rightarrow F, T_1$	$T_0.n := F.n$ **and** $T_1.n$; $T_0.e := F.e$ **and** $T_1.e$; **if** $F.w = \emptyset$ **or** $F.w = \lambda$ **then** $T_0.w := T_1.w$ **else if** $T_1.w = \emptyset$ **or** $T_1.w = \lambda$ **then** $T_0.w := F.w$ **else begin** $T_0.w := F.w \|\|$",$\|\|T_1.w$; $T_0.n :=$ FALSE **end** **endif** **endif**
$T \rightarrow F$	$T.w := F.w$; $T.n := F.n$; $T.e := F.e$

Figure 2: *Syntax-directed definition for the regular expression simplification process (part 1). All attributes are synthetic.*

PRODUCTION	TRANSLATION
$F \rightarrow W$	$F.w := W.w$; $F.n := W.n$; $F.e := W.e$
$F \rightarrow W*$	$F.n :=$ TRUE; $F.e :=$ TRUE; **if** $W.w = \emptyset$ **or** $W.w = \lambda$ **then** $F.w := \lambda$ **else** $F.w := W.w \|\|$"$*$" **endif**
$F \rightarrow W+$	$F.n :=$ TRUE; $F.e := W.e$; **if** $W.w = \emptyset$ **or** $W.w = \lambda$ **then** $F.w := W.w$ **else** $F.w := W.w \|\|$"$+$" **endif**
$F \rightarrow W?$	$F.n :=$ TRUE; $F.e :=$ TRUE; **if** $W.w = \emptyset$ **or** $W.w = \lambda$ **then** $F.w := \lambda$ **else** $F.w := W.w \|\|$"?" **endif**
$W \rightarrow$ **name**	$W.n :=$ TRUE; $W.e :=$ FALSE; **if** HasWitnesses(**name**.w) **then** $W.w :=$ **name**.w **else** $W.w := \emptyset$ **endif**
$W \rightarrow (E)$	W.n := E.n; W.e := E.e; **if** $E.w = \emptyset$ **or** $E.w = \lambda$ **then** $W.w := E.w$ **else if** $E.n$ **then** $W.w := E.w$ **else begin** $W.w :=$ "("$\|\|E.w\|\|$")"; $W.n :=$ TRUE **end** **endif** **endif**

Figure 3: *Syntax-directed definition for the regular expression simplification process (part 2). All attributes are synthetic.*