# Identifying a reduced DTD from marked up documents*

Alejandro Bia†, Rafael C. Carrasco‡ and Mikel L. Forcada‡
†Miguel de Cervantes Digital Library. Universidad de Alicante. Spain
‡Dept. de Lenguajes y Sistemas Informáticos. Universidad de Alicante. Spain.

* Work supported by the Spanish CICyT through grant TIC97–0941

# Identifying a reduced DTD from marked up documents*

**Abstract**

This paper describes a method for the automatic generation of simplified DTDs from a source DTD and a sample of marked up documents. The purpose is to create the minimal DTD with which all the documents in the the sample comply. In this way, new files can be created and parsed using this simplified DTD but still being compliant with the original, more general one. The pruned DTD makes the task of markup easier, specially for non-experienced XML writers.

This tool was used to obtain simplified versions of the Text Encoding Initiative DTD to be used at the Miguel de Cervantes digital library[1]. This work is part of a larger project in the field of text markup and derived applications [1].

*Keywords*: automatic learning, feature extraction, grammatical inference, document analysis, document markup, digital libraries.

## 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 side of the productions are unambiguous regular expressions [2]. Previous work has addressed the task of identifying a DTD from examples. A common difficulty in this approach is the need to find a correct degree of generalization. Some practical tools as FRED [3] let the users customize their preferred degree of generalization. Ahonen [4, 5] builds a $(k, h)$-testable model for the element contents and needs non-trivial further generalization in order to disambiguate the model [6].

---

Work supported by the Spanish CICyT through grant TIC97–0941

[1] http://cervantesvirtual.com/

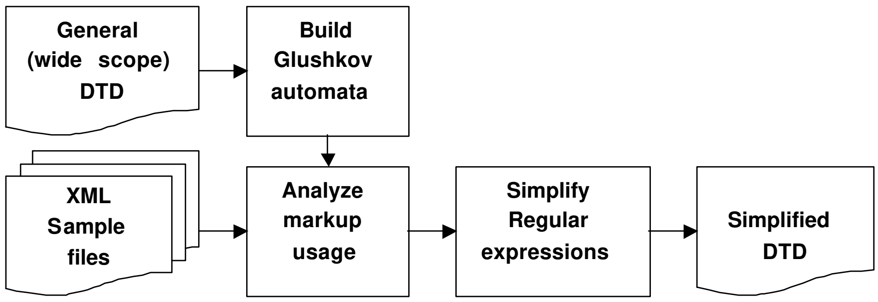


Figure 1: *Architecture of the DTD simplifier*

Young-Lai and Tompa [7] rely on a stochastic approach to control overgeneralization, based in turn on the algorithm by Carrasco and Oncina [8]. Presumably, the stochastic approach needs large collections of hand-tagged documents. Pizza-Chef [9] is a tool to generate DTDs suited to a collection of particular tasks and compliant with the markup directives defined by the Text Encoding Initiative (TEI).

However, a general DTD defining a global frame that a whole set of files must fulfill allows for a natural way to avoid overgeneralization. Indeed, any particularized, narrow-scope DTD should not accept any document that is not accepted by the general, wide-scope one.

Therefore, the objective of our approach is to automatically select only those DTD features that are used by a set of valid 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.

Using this automated method, 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.

This technique also allows us to build a one-document DTD, i.e. the minimal markup schema derived from the general DTD that a given XML document complies. 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 elements within others helps us to detect unusual structures that could reflect markup mistakes or DTD features that allow for unwanted generalization.

## 2 Motivation and general description

Saving the cost of developing our own DTD and text interchangeability are some of the reasons why the `teixlite.dtd`[2], XML version of the SGML `teilite.dtd` of the TEI scheme, has been chosen at the Miguel de Cervantes Digital Library. But the `teixlite.dtd` is still too complex for markup beginners. Our markup team is composed mostly of humanists with some computer skills who appreciate their computer work be simplified as much as possible.

On the other hand our XML documents do not use, and do not need all the markup options provided by the `teixlite.dtd`. So a simpler DTD was needed to simplify markup tasks and to avoid possible use of unwanted markup options. But we still wanted our files to be TEI compliant and benefit from the advantages of sharing a common DTD with other international digitization projects.

We started by defining what kinds of modifications will 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 is clear that doing the simplifications by hand is tedious and error prone. Constructing a set of sample documents representative of all the types of documents we need to markup together with a program that simplifies the DTD automatically will alleviate this task.

A diagram of the process is shown in figure 1. As the diagram shows, the general DTD is processed to extract the structure of the markup models and a Glushkov automaton [10] is built for each one (that is, for each regular expression). The XML sample files are then preprocessed to extract the elements used and their nesting

[2] Available through the TEI consortium at `http://www.tei-c.org`.

patterns. We keep track of the elements used in the sample files and mark the visited states of the automata. Finally, we eliminate unused elements and simplify the right parts of element definitions, i.e. the regular expressions that define further nestings.

For the implementation of the DTD prune toolkit we needed both an XML and a DTD parser. We assumed that both the XML sample files and the source DTD would be well-formed and valid, so there would be no need to build validating parsers. In particular, regular expressions are parsed against the EBNF grammar described in the following section, although indeed, XML forces stricter parentization patterns.

## 3 Theoretical foundation

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

$$\begin{aligned} &E \to T|T\text{“}|\text{”}E \\ &T \to F|F\text{“},\text{”}T \\ &F \to W|W\text{“}*\text{”}|W\text{“}+\text{”}|W\text{“?”} \\ &W \to \text{“(”}E\text{“)”} \\ &W \to a_1|a_2|...|a_{|\Sigma|} \end{aligned} \tag{1}$$

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

A *marking* of $r$ is a pair $(\Phi_r, E_r)$ with

- $E_r \in \mathrm{reg}(\mathbb{N})$ such that no $n \in \mathbb{N}$ is used in $E_r$ more than once;
- $\Phi_r : \mathbb{N} \to \Sigma$ 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 $\Phi_r$ to work on subexpressions of $E_r$ if we assume that $\Phi_r$ 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 look-ahead by at most one token in the regular expression. 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

$n, m \in \mathbb{N}$

$$\left.\begin{array}{c} xny \in L(E_r) \\ xmz \in L(E_r) \\ n \neq m \end{array}\right\} \Rightarrow \Phi_r(xny) \neq \Phi_r(xmy) \tag{2}$$

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

**Theorem 1** *(Lemma 2.5 in [2]) 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 [10]. 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, $L(f(E_r)) = L(E_r) \cap \mathrm{reg}(\mu)$ and then $L(f(E_r)) \subseteq L(E_r)$. Therefore, we may substitute $L(E_r)$ by $L(f(E_r))$ in definition (2) and the implication remains valid. Then, $\Phi_r(f(E_r))$ is 1-unambiguous.

# 4 Regular expression pruning

The process by means of which each regular expression is simplified is based on a bottom-up parse of the original regular expression. A syntax-directed definition [11] is shown in an appendix. 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 $\mathrm{reg}(\Sigma)$ space; finally, the resulting regular expression is rearranged to avoid using symbols not in $\Sigma$.

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

$$\begin{array}{rcccl} \emptyset, E &=& E, \emptyset &=& \emptyset + = \emptyset \\ \emptyset | E &=& E | \emptyset &=& E \\ \emptyset * &=& \emptyset ? &=& \lambda \end{array} \qquad \begin{array}{rcccl} \lambda, E &=& E, \lambda &=& E \\ \lambda | E &=& E | \lambda &=& \left\{ \begin{array}{cl} E & \text{if empty}(E) \\ E? & \text{otherwise} \end{array} \right. \\ \lambda * &=& \lambda + &=& \lambda ? = \lambda \end{array} \tag{3}$$

where $\lambda$ 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 [10]).

## 5 Conclusions and future work

We have developed a method which has been used to automatically generate simplified DTDs at the Miguel de Cervantes Digital Library. On this first stage, we addressed the simplification of element type descriptions based on sample files. On a second stage, we plan to add the automatic elimination or addition of attributes. We also plan to collect statistics to detect unusual patterns that may reflect markup mistakes.

## References


[1] Alejandro Bia and Andrés Pedreño. The Miguel de Cervantes Digital Library: The Hispanic Voice on the WEB. *Literary and Linguistic Computing Journal*, (to appear) 2000.

[2] Anne Brüggemann-Klein and Derick Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.

[3] Keith 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.

[4] Helena Ahonen. Automatic generation of SGML content models. *Electronic Publishing Origination, Dissemination, and Design*, 8(2/3):195–206, June/September 1995.

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

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

[7] Matthew Young-Lai and Frank W. M. Tompa. Stochastic grammatical inference of text database structure. *Machine Learning*, 40(2):1, 2000.

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

[9] Lou Burnard. The Pizza Chef: a TEI Tag Set Selector. http://www.hcu.ox.ac.uk/TEI/pizza.html, September 1997.

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

[11] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers Principles, Techniques, and Tools*. Addison Wesley, 1986.

# A Building the Glushkov automata of an expression

Given a regular expression $r$ we describe the procedure [10] 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 empty : reg($\mathbb{N}$) $\to$ boolean is true if the subexpression contains the empty string and can be computed as

$$\begin{array}{rcl}
\mathrm{empty}(n) & = & \mathrm{FALSE} \\
\mathrm{empty}(F|G) & = & \mathrm{empty}(F) \vee \mathrm{empty}(G) \\
\mathrm{empty}(F,G) & = & \mathrm{empty}(F) \wedge \mathrm{empty}(G) \\
\mathrm{empty}(F*) & = & \mathrm{TRUE} \\
\mathrm{empty}(F+) & = & \mathrm{empty}(F) \\
\mathrm{empty}(F?) & = & \mathrm{TRUE}
\end{array} \tag{4}$$

The subset first : 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

$$\begin{array}{rcl}
\mathrm{first}(n) & = & \{n\} \\
\mathrm{first}(F|G) & = & \mathrm{first}(F) \cup \mathrm{first}(G) \\
\mathrm{first}(F,G) & = & \begin{cases} \mathrm{first}(F) \cup \mathrm{first}(G) & \text{if } \mathrm{empty}(F) \\ \mathrm{first}(F) & \text{otherwise} \end{cases} \\
\mathrm{first}(F*) & = & \mathrm{first}(F) \\
\mathrm{first}(F+) & = & \mathrm{first}(F) \\
\mathrm{first}(F?) & = & \mathrm{first}(F)
\end{array} \tag{5}$$

The subset last : 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

$$\begin{array}{rcl}
\mathrm{last}(n) & = & \{n\} \\
\mathrm{last}(F|G) & = & \mathrm{last}(F) \cup \mathrm{last}(G) \\
\mathrm{last}(F,G) & = & \begin{cases} \mathrm{last}(F) \cup \mathrm{last}(G) & \text{if } \mathrm{empty}(G) \\ \mathrm{last}(G) & \text{otherwise} \end{cases} \\
\mathrm{last}(F*) & = & \mathrm{last}(F) \\
\mathrm{last}(F+) & = & \mathrm{last}(F) \\
\mathrm{last}(F?) & = & \mathrm{last}(F)
\end{array} \tag{6}$$

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

$$\begin{array}{rcl} \mathrm{follow}(n) & = & \emptyset \\ \mathrm{follow}(F|G) & = & \mathrm{follow}(F) \cup \mathrm{follow}(G) \\ \mathrm{follow}(F,G) & = & \mathrm{follow}(F) \cup \mathrm{follow}(G) \cup \mathrm{last}(F) \times \mathrm{first}(G) \\ \mathrm{follow}(F*) & = & \mathrm{follow}(F) \cup \mathrm{last}(F) \times \mathrm{first}(F) \\ \mathrm{follow}(F+) & = & \mathrm{follow}(F) \cup \mathrm{last}(F) \times \mathrm{first}(F) \\ \mathrm{follow}(F?) & = & \mathrm{follow}(F) \end{array} \tag{7}$$

With the above elements, the Glushkov automaton [2] $(Q, \mathbb{N}, \delta, I, F)$ is given by

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

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

# B Syntax-directed definition

The simplified regular expression is stored in attribute $.s$, whereas attributes $.n$ and $.e$ are auxiliary boolean attributes which store if the subexpression is simple and accepts the empty string respectively

<table>
<tr><th>PRODUCTION</th><th>TRANSLATION</th></tr>
<tr><td>$E_0 \to T|E_1$</td><td>$E_0.n := T.n$ and $E_1.n; E_0.e := T.e$ or $E_1.e$;<br>if $T.w = \emptyset$<br>then $E_0.w := E_1.w$<br>else if $T.w = \lambda$<br>then if $E_1.w = \emptyset$ or $T.w = \lambda$ or $E_1.e$<br>then $E_0.w := E_1.w$<br>else $E_0.w := E_1.w||$"?"<br>endif<br>else if $E_1.w = \lambda$<br>then if $T.e$<br>then $E_0.w := T.w$<br>else $E_0.w := T.w||$"?"<br>endif<br>else if $E_1.w \neq \emptyset$<br>then begin $E_0.w := T.w||$"|"$||E_1.w$;<br>$E_1.n :=$ FALSE<br>end<br>endif<br>endif<br>endif<br>endif</td></tr>
<tr><td>$E \to T$</td><td>$E.w := T.w; E.n := T.n; E.e := T.e$</td></tr>
<tr><td>$T_0 \to F, T_1$</td><td>$T_0.n := F.n$ and $T_1.n; T_0.e := F.e$ and $T_1.e$;<br>if $F.w = \emptyset$ or $F.w = \lambda$<br>then $T_0.w := T_1.w$<br>else if $T_1.w = \emptyset$ or $T_1.w = \lambda$<br>then $T_0.w := F.w$<br>else begin<br>$T_0.w := F.w||$","$||T_1.w$;<br>$T_0.n :=$ FALSE<br>end<br>endif<br>endif</td></tr>
<tr><td>$T \to F$</td><td>$T.w := F.w; T.n := F.n; T.e := F.e$</td></tr>
</table>

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

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

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