Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

ELSEVIER

Information Processing Letters 85 (2003) 7–12

Information Processing Letters

www.elsevier.com/locate/ipl

# Efficient extraction of schemas for XML documents

Jun-Ki Min, Jae-Yong Ahn, Chin-Wan Chung *

*Division of Computer Science, Department of Electrical Engineering & Computer Science, Korea Advanced Institute of Science and Technology, 373-1, Kusong-dong, Yusong-gu, Taejon, 305-701, Republic of Korea*

Received 5 February 2002; received in revised form 31 May 2002

Communicated by K. Iwama

**Abstract**

In this paper, we present a technique for efficient extraction of concise and accurate schemas for XML documents. By restricting the schema form and applying some heuristic rules, we achieve the efficiency and conciseness. The result of an experiment with real-life DTDs shows that our approach attains high accuracy and is 20 to 200 times faster than existing approaches.


*Keywords:* XML; Automatic schema extraction; DTD; XML schema; Databases

## 1. Introduction

There has been an increasing interest in XML (eXtensible Markup Language) [4], since it is spotlighted as the standard for data representation and exchange in the Web. To describe the schema (i.e., the summary of structure) of XML documents, many languages such as Document Type Definition (DTD) [4], Document Content Description (DCD) [3], and XML Schema [6] have been proposed.

The schema for XML documents serves several important purposes as follows:

- To enable the designer to describe the structure of XML documents.
- To transform XML documents to different types of data such as relational data.
- To enable the efficient XML query processing such as query pruning and rewriting.

In spite of its importance, unfortunately, the schema is not mandatory for XML documents and many XML documents in the Web do not have the accompanying schemas. These XML documents cannot take advantages of the schema. Therefore, automatic schema extraction tools have been introduced such as XTRACT [7], DDbE [2], and DTD-Miner [8]. However, the quality of the schema inferred by some tools is poor and some tools consume too much time to get the result.

**Our contributions.** In this paper, we describe our approach to an efficient extraction of schemas for XML documents. To achieve the efficiency, we devise a restricted representation form (i.e., *restricted element content model*) of the schema and make some heuristic

* Corresponding author.
*E-mail addresses:* jkmin@islab.kaist.ac.kr (J.-K. Min), jyahn@islab.kaist.ac.kr (J.-Y. Ahn), chungcw@islab.kaist.ac.kr (C.-W. Chung).

0020-0190/02/$ – see front matter 
PII: S0020-0190(02)00345-9

rules for the schema generation. Our restricted element content model is transformed into DTD, directly. Also, it can be transformed into simplified XML Schema which does not contain some information such as inheritance, polymorphism, and name space information.

## 2. Problem statement

Generally, the languages specifying schemas for XML documents, such as DTD and XML Schema, describe the structure of an element by specifying the regular expression that its subelement (i.e., child-level element) sequences have to conform to. The generation of the regular expression from examples has been well studied in the field of machine learning [1,5]. However, Angluin [1] showed that the whole class of regular languages cannot be polynomially identified in the limit of the representation of deterministic finite automata (DFA). Also, without prior information, it is very difficult to find the intention of the XML document generator. Consider the XML document in Fig. 1.

Intuitively, naive schema information for *author* element in Fig. 1 is a simple description of all subelement names, that is (first|last)*. Although this approach suggests concise schema information, the order information among siblings is lost. On the other hand, the most accurate schema information is to merge all subelement sequences with or (|) operator, that is ((first last)|(last first)). However, this approach tends to be voluminous. As the *conciseness* and the *accuracy* are contradicting each other, finding the right balance is a difficult task. Thus, our goal is to efficiently derive a concise and accurate regular expression that subelement sequences for an element conform to.

```
⟨book⟩
  ⟨title⟩ title1 ⟨/title⟩
  ⟨author⟩
    ⟨first⟩ first1 ⟨/first⟩ ⟨last⟩ last1 ⟨/last⟩
  ⟨/author⟩
  ⟨author⟩
    ⟨last⟩ last2 ⟨/last⟩ ⟨first⟩ first2 ⟨/first⟩
  ⟨/author⟩
⟨/book⟩
```

Fig. 1. An example of XML document.

In this paper, we do not consider the problem of inferring attribute information of an element and that of the derivation of the primitive type (e.g., integer, float, string) for the value of an element, since they are straightforward tasks.

## 3. Schema extraction

### 3.1. Restricted element content model

As mentioned early, many researchers already have recognized the difficulties of inferring the regular expression from examples. Like most related literature, we restrict the form of the regular expression. The element content model [4] describes the content of an element in XML documents. First of all, for simplicity, we encode each tag name (i.e., an element) that appears in XML documents to a unique symbol. Also, to extract the content information efficiently, we restrict the element content model as follows.

**Definition 1** (*Element Content Model*).

(Element Content Model)
$E := (T_1 \dots T_k)^{\langle min,max\rangle}$

(Term)
$T_i := \left(s_{i1}^{opt} \dots s_{ij}^{opt}\right)^{\langle min,max\rangle}$ //sequence of symbols
or $\left(s_{i1}^{opt} | \dots | s_{ij}^{opt}\right)^{\langle min,max\rangle}$ //choice of symbols

where $min = 0$ *or* 1, $max \geqslant 1$, and $opt =$ true *or* false.

Our element content model can be transformed into both DTD and XML Schema since the model keeps the occurrence information using *min* and *max*, and correlated symbols are united as a *term* which can be naturally transformed into a complex type in the XML Schema. The model keeps the maximum occurrence information using *max* and the optional or mandatory occurrence of a term or a content using *min*. However, note that, our element content model does not contain inheritance, polymorphism, and namespace information for XML Schema since the derivation of these information from XML document is very difficult and impossible in some cases. The *opt* flag of a symbol denotes the optional representation of the symbol in a term (i.e., $opt =$ false means mandatory). For brevity,

we omit *opt*, *min* and *max* if *opt* is false, and *min* and *max* are both 1.

For example, a regular expression $(a\ b^*\ c?)^*$ can be represented as $E = (T_1\ T_2\ T_3)^{\langle 0,\infty\rangle}$, where $T_1 = (a)$, $T_2 = (b)^{\langle 0,\infty\rangle}$, and $T_3 = (c^{opt})$ by using our element content model. However, our element content model cannot represent all kinds of regular expressions. For example, regular expressions such as $(a\ (b|c+)\ d)^*$ cannot be represented by our model, since $(b|c+)$ cannot be represented in a term. Instead, with a little loss of accuracy, it can be represented as $((a)\ (b|c)^{\langle 1,\infty\rangle}\ (d))^{\langle 0,\infty\rangle}$.

To achieve the *conciseness*, we enforce the *No Duplication Type* property to the element content model.

**Property 1** (*No Duplication Type*). Let an element content model $E$ be $(T_1 \ldots T_k)^{\langle min,max\rangle}$ and

$$\sigma(T_x) = \{s_{xy} \mid s_{xy} \text{ is a symbol in } T_x \text{ which is a term in } E\}.$$

*Disjoint Term*: If $i \neq j$ for $1 \leqslant i, j \leqslant k$, then $\sigma(T_i) \cap \sigma(T_j) = \emptyset$.

*Disjoint Symbol*: For each term $T_l$ in $E$ such that $\sigma(T_l) = \{s_{l1} \ldots s_{ln}\}$, if $a \neq b$ for $1 \leqslant a, b \leqslant n$, then $s_{la} \neq s_{lb}$.

The informal description of Property 1 is that the same symbol must not appear more than once in an element content model. Even though Property 1 is a validity constraint for the *mixed content* in DTD, according to our analysis of real-life DTDs and XML Schemas in [9], most of DTDs and XML Schemas satisfy Property 1. Also, DTD and XML Schema impose that the element content model should be deterministic, that is, a subelement in a subelement sequence can be validated using DTD or XML Schema without looking ahead (i.e., the 1-unambiguity constraint). Property 1 is a sufficient condition of the 1-unambiguity imposed by DTD and XML Schema.

### 3.2. Extraction of schema for an element

Our schema extraction algorithm consists of several important steps as shown in Fig. 2. We apply bottom-up approach. An element content model is made for each subelement sequence of element *e*, and then all element content models are consolidated for element *e*. For example, the element content model for *book* in Fig. 1 is (title author$^{\langle 1,2\rangle}$), whereas that for *author* is (first|last)$^{\langle 1,2\rangle}$ which is consolidated from two models (first last) and (last first).

**Algorithm**

1. Collect all subelement sequences of element $e$ into a set $I$
2. Extract an element content model $E_I$ for $e$
   2.1. *Partition* each sequence in $I$
   2.2. *Infer* an element content model for each sequence
   2.3. *Consolidate* all element content models into $E_I$
3. Translate $E_I$ to DTD form or XML Schema form

Fig. 2. Procedural overview.

We start by partitioning an input sequence. In the partitioning step, a sequence $N$ is decomposed into subsequences $N_1, N_2, \ldots, N_k$. A subsequence $N_i$ has following properties: a symbol in $N_i$ does not appear more than once unless it appears consecutively and a symbol appearing consecutively such as *aaaa* is a subsequence $N_i$ with only that symbol, where $N_i$ keeps the repeating number of the symbol such as $(a)^{\langle 1,4\rangle}$.

**Example 1.** Given a sequence *abcbcddefefddgggabcbc*, the result of partitioning is:

$$(abc), (bc), (d)^{\langle 1,2\rangle}, (ef), (ef), (d)^{\langle 1,2\rangle}, (g)^{\langle 1,3\rangle}, (abc), (bc).$$

Note that the condition of whether a symbol in $N_i$ occurs only once can be checked in $O(d)$ where $d$ is the number of distinct symbols in $N$. Therefore, the time complexity of the partitioning procedure can be easily shown to be $O(d|N|)$ where $|N|$ is the length of a sequence $N$.

The second step is the inference of the element content model for a single input sequence using the result of the partitioning procedure. The main role of this step is to fold the subsequences having repeating parts using the maximum number of repeat. Considering Example 1, $(abc)$ and $(bc)$ are merged into $(a)(bc)^{\langle 1,2\rangle}$.

First, we show some heuristic rules followed by an example for folding a repeating part $N_a$ of a subsequence into a term $T_i$ in an element content

model $E$. Note that, to obtain a larger number of repeats for $T_i$ and $N_a$, we use the max function when *max* of $E$ is greater than 1.

**Folding Rules** $fold(T_i, N_a)$.
Assume that $T_i$ is a sequence of symbols (called *seq*-term), $(s_{i1}^{opt} \dots s_{ij}^{opt})^{\langle n,x\rangle}$ or a choice of symbols (called *or*-term), $(s_{i1}^{opt} | \dots | s_{ij}^{opt})^{\langle n,x\rangle}$, and $N_a = (n_1 \dots n_b)^{\langle 1,r\rangle}$.

**F1.** If $\sigma(N_a) \not\subseteq \sigma(T_i)$, $fold(T_i, N_a)$ is *undefined*.

**F2.** Let $T_i$ be an *or*-term.

$$fold(T_i, N_a) = \left(s_{i1}^{opt} | \dots | s_{ij}^{opt}\right)^{\langle n,x'\rangle},$$

where if *max* of $E$ is 1, then $x' = x + rb$ else $x' = \max(x, rb)$ since a symbol in $N_a$ is repeated at most $rb$ times.

**F3.** Let $T_i$ be a *seq*-term. If *max* of $E$ is 1 and $x$ is 1, and there is $(s_{i1}^{opt} \dots s_{ik}^{opt})$ such that

$$\sigma\left(s_{i1}^{opt} \dots s_{ik}^{opt}\right) \cap \sigma(N_a) = \emptyset,$$

$$fold(T_i, N_a) = \left(s_{i1}^{opt} \dots s_{ik}^{opt}\right) fold\left((s_{ik+1}^{opt} \dots s_{ij}^{opt})^{\langle n,1\rangle}, N_a\right).$$

**F4.** Let $T_i$ be a *seq*-term. If *max* of $E$ is greater than 1 or $x > 1$ or $s_{i1} \in \sigma(N_a)$, and, for $\forall n_m \in \sigma(N_a)$, if $n_m = s_{ik}$ then $n_{m+1} = s_{il}$ for $k < l$,

$$fold(T_i, N_a) = \left(s_{i1}^{opt|optional_1} \dots s_{ij}^{opt|optional_j}\right)^{\langle n,x'\rangle},$$

where if $s_{ip} \notin \sigma(N_a)$ for $1 \leqslant p \leqslant j$, then $optional_p$ = true else $optional_p$ = false, and if *max* of $E$ is 1, then $x' = x + r$ else $x' = \max(x, r)$.

**F5.** Let $T_i$ be a *seq*-term. If *max* of $E$ is greater than 1 or $x > 1$ or $s_{i1} \in \sigma(N_a)$, and $\exists n_m, n_q \in \sigma(N_a)$ such that $n_m = s_{ik}$ and $n_q = s_{il}$ for $k > l$ and $m < q$,

$$fold(T_i, N_a) = \left(s_{i1}^{opt} | \dots | s_{ij}^{opt}\right)^{\langle n,x'\rangle},$$

where if *max* of $E$ is 1, then $x' = jx + rb$ else $x' = \max(jx, rb)$.

Example 2 shows some representative applications of *Folding Rules*.

**Example 2.** Suppose *max* of $E$ is 1.

(1) Application of F2:

$$fold\left((a|b|c)^{\langle 1,5\rangle}, (bc)\right) = (a|b|c)^{\langle 1,7\rangle}.$$

(2) Application of F3:

$$fold\left((abcd), (bc)\right) = (a) fold\left((bcd), (bc)\right).$$

(3) Application of F4:

$$fold\left((abcd)^{\langle 1,5\rangle}, (bc)\right) = (a^{opt} bcd^{opt})^{\langle 1,6\rangle}.$$

(4) Application of F5:

$$fold\left((ab)^{\langle 1,2\rangle}, (ba)\right) = (a|b)^{\langle 1,6\rangle}.$$

However, applying just above rules cannot preserve the *disjoint term* condition of Property 1. Considering Example 1, $(abc), (bc), (d)^{\langle 1,2\rangle}, (ef), (ef)$ fold into an element content model $E = (T_1\ T_2\ T_3\ T_4)^{\langle 1,1\rangle}$ where $T_1 = (a)^{\langle 1,1\rangle}$, $T_2 = (bc)^{\langle 1,2\rangle}$, $T_3 = (d)^{\langle 1,2\rangle}$, and $T_4 = (ef)^{\langle 1,2\rangle}$. Subsequently, we try to insert the next subsequence $(d)^{\langle 1,2\rangle}$ into $E$. However, the symbol $d$ already appeared in $T_3$. In this case, there are alternatives to insert $(d)^{\langle 1,2\rangle}$ into $E$. One is making an *or*-term such as $((a)^{\langle 1,1\rangle}(bc)^{\langle 1,2\rangle}(d|e|f)^{\langle 1,8\rangle})^{\langle 1,1\rangle}$. The other is making optional terms such as $((a)^{\langle 0,1\rangle}(bc)^{\langle 0,2\rangle}(d)^{\langle 1,2\rangle}(ef)^{\langle 0,2\rangle})^{\langle 1,2\rangle}$. Generally, symbols in an *or*-term appear closely. Thus, based on the locality, we choose one from the two alternatives according to a parameter `ThresHold`. The following rules describe how to merge or insert a subsequence $N_a$ into an element content model $E$.

**Relaxed Transformation Rules**
Assuming that $E$ is $(T_1 \dots T_j \dots T_k)^{\langle min,max\rangle}$, we try to merge $N_a$ into $T_j$ which was inserted or folded at the previous step. Let $N'_a$ be a prefix of $N_a$ such that all symbols in $N'_a$ are in some $T_i$ of $E$ or none of the symbols in $N'_a$ appears in $E$.

**R1.** If none of the symbols in $N'_a$ appears in $E$, $N'_a$ is inserted at the $(j+1)$th position as $T_{j+1}$. In this case, if *max* of $E$ is greater than 1, *min* of $T_{j+1}$ is set to 0 because $N'_a$ did not appear in the previous steps.

**R2.** If $0 < j - i <$ `ThresHold`, $T_i \dots T_j$ are replaced by a new $T_i = (s_{i1}| \dots |s_{im})^{\langle n,x\rangle}$ where $\bigcup_{l=i}^{j} \sigma(T_l) = \sigma(\text{new } T_i)$, $n = \prod_{l=i}^{j} (min \text{ of } T_l)$, and $x = \sum_{l=i}^{j} (max \text{ of } T_l \cdot \text{number of symbols in } T_l)$.

**R3.** If $j - i \geqslant$ `ThresHold`, *min*'s of $T_1 \dots T_{i-1}$ and $T_{j+1} \dots T_k$ are set to 0 and *max* of $E$ is increased by 1.

**R4.** If $j - i \leqslant 0$, *min*'s of $T_{j+1} \ldots T_{i-1}$ are set to 0.

For R2, R3, and R4, $fold(T_i, N'_a)$ is applied. And then, the above rules are applied for the remainder of $N_a$, repeatedly.

Finally, if there is no more subsequence, *min*'s of $T_{j+1} \ldots T_k$ are set to 0.

The following example shows how to apply *Relaxed Transformation Rules* and *Folding Rules* on Example 1.

**Example 3.** Given subsequences, $(abc)$, $(bc)$, $(d)^{\langle 1,2\rangle}$, $(ef)$, $(ef)$, $(d)^{\langle 1,2\rangle}$, $(g)^{\langle 1,3\rangle}$, $(abc)$, $(bc)$ and `ThresHold` 2, an element content model $E$ is obtained as follows:

Insert $(abc)$: $E = ((abc))$
  // *by applying* R1
Insert $(bc)$: $E = ((a)(bc)^{\langle 1,2\rangle})$
  // R4, F3, F4
Insert $(d)^{\langle 1,2\rangle}$: $E = ((a)(bc)^{\langle 1,2\rangle}(d)^{\langle 1,2\rangle})$
  // R1
Insert $(ef)$: $E = ((a)(bc)^{\langle 1,2\rangle}(d)^{\langle 1,2\rangle}(ef))$
  // R1
Insert $(ef)$: $E = ((a)(bc)^{\langle 1,2\rangle}(d)^{\langle 1,2\rangle}(ef)^{\langle 1,2\rangle})$
  // R4, F4
Insert $(d)^{\langle 1,2\rangle}$: $E = ((a)(bc)^{\langle 1,2\rangle}(d|e|f)^{\langle 1,8\rangle})$
  // R2, F2
Insert $(g)^{\langle 1,3\rangle}$: $E = ((a)(bc)^{\langle 1,2\rangle}(d|e|f)^{\langle 1,8\rangle}(g)^{\langle 1,3\rangle})$
  // R1
Insert $(abc)$: $E = ((a)(bc)^{\langle 1,2\rangle}(d|e|f)^{\langle 1,8\rangle}(g)^{\langle 1,3\rangle})^{\langle 1,2\rangle}$
  // R3, F3, R5 and F4, since $(abc)$ is divided into $(a)(bc)$
Insert $(bc)$: $E = ((a)(bc)^{\langle 1,2\rangle}(d|e|f)^{\langle 1,8\rangle}(g)^{\langle 1,3\rangle})^{\langle 1,2\rangle}$
  // R4, F4
Finally, $E = ((a)(bc)^{\langle 1,2\rangle}(d|e|f)^{\langle 0,8\rangle}(g)^{\langle 0,3\rangle})^{\langle 1,2\rangle}$
The DTD form of $E$ is $((a)(bc)+(d|e|f)*(g)*)+$

Next, we consolidate all element content models into the final element content model $E_I$ by factoring and *or*-ing. Intuitively, we greedily factor shared common prefix or suffix terms and merge remainders by *or*-ing. For example, for element content models $((a)^{\langle n_1,x_1\rangle}(b)^{\langle n_2,x_2\rangle}(d)^{\langle n_3,x_3\rangle})^{\langle n_4,x_4\rangle}$ and $((a)^{\langle n_5,x_5\rangle}(c)^{\langle n_6,x_6\rangle}(d)^{\langle n_7,x_7\rangle})^{\langle n_8,x_8\rangle}$, the consolidation step generates

$$\big((a)^{\langle \mathrm{MIN}(n_1,n_5),\mathrm{MAX}(x_1,x_5)\rangle} (b|c)^{\langle \mathrm{MIN}(n_2,n_6),\mathrm{MAX}(x_2,x_6)\rangle} (d)^{\langle \mathrm{MIN}(n_3,n_7),\mathrm{MAX}(x_3,x_7)\rangle}\big)^{\langle \mathrm{MIN}(n_4,n_8),\mathrm{MAX}(x_4,x_8)\rangle}.$$

Due to the space constraints, we omit the detailed behavior (which is a simplification of the factoring algorithm in XTRACT [7]).

## 4. Experiment

To show the efficiency and accuracy, we compare the resulting DTDs of our approach with XTRACT and DDbE. Since the detailed mechanism of DTD-Miner was not described in [8], we did not include DTD-Miner in the experimental study. For the experiment, we implemented our approach in Java and used the XML4J parser[1] to parse XML documents. XTRACT was originally written in C++ and, therefore, we had to implement it in Java to make a fair comparison in view of efficiency. During the implementation of XTRACT in Java, we simplified the MDL Subsystem of XTRACT using the greedy method instead of the Facility Location Problem (FLP) approximation. Thus, our implementation of XTRACT may consume less time than the Java implementation of the original XTRACT algorithm.

As the dataset, we used the real-life DTDs which were used in the experiment of XTRACT [7]. The original DTDs are shown in the second column of Table 1. In order to evaluate the accuracy of DTD, we generated 1000 elements for each DTD using the XML Generator from IBM.[2] The experiment was performed on Pentium III-866 MHz platform with MS-Windows XP and 256 MBytes of main memory.

In Table 1, we show the obtained DTD for each dataset consisting of 1000 elements. The resulting DTDs of XTRACT are obtained from [7][3] and other DTDs are obtained by our experiment. Since we used DDbE version 2, some DTDs obtained by DDbE are different from these reflect in [7]. As shown in Table 1, our approach generates the same or better DTDs compared with XTRACT. DDbE usually generates too complex results and could not generate the fifth DTD.

[1] Available at http://www.alpahworks.ibm/tech/xml4j.
[2] Available at http://www.alpahworks.ibm/tech/xmlgenerator.
[3] The sixth DTD was not reported in [7].

Table 1
Generated DTDs

| No. | Original DTD | Our approach | XTRACT | DDbE ver2 |
|---|---|---|---|---|
| 1 | $a\|b\|c\|d\|e$ | $(a\|b\|c\|d\|e)$ | $a\|b\|c\|d\|e$ | $(a\|b\|c\|d\|e)$ |
| 2 | $(a\|b\|c\|d\|e)^*$ | $(a\|b\|c\|d\|e)^*$ | $(a\|b\|c\|d\|e)^*$ | $((a\|(e\|b\|c\|a\|d\|(d+c))\|(e\|a\|(e\|c\|d\|(e+b+))\|c\|b\|(e+b+)$ $\|d\|(d+c))\|d\|e\|b\|c\|(e+b+)\|(dcb)\|(d+c))^*)$ |
| 3 | $ab^*c^*$ | $(ab^*c^*)$ | $(ab^*c^*)$ | $(a(b\|c)+)$ |
| 4 | $a^*b?c?d?$ | $(a^*b?c?d?)$ | $a^*b?c?d?$ | $(((a\|b)\|c\|d\|a\|((b\|c)\|d\|b\|c)\|b)+)$ |
| 5 | $(a(bc)+d)^*$ | $(a(bc)+d)^*$ | $(a(bc)^*d)^*$ | – |
| 6 | $(ab?c^*d?)^*$ | $(a(b\|c\|d)^*)^*$ | – | $((((ac+ac+d)\|(a+b)\|a)\|\ldots\|(c+dab))^*)$ |

Table 2
DTD generation time (sec)

| No. | Our approach | Simple XTRACT | DDbE ver2 |
|---|---|---|---|
| 1 | 0.4 | 25.8 | 228 |
| 2 | 0.65 | 25.6 | 229 |
| 3 | 1.27 | 25.0 | 230 |
| 4 | 0.5 | 24.1 | 231 |
| 5 | 2.33 | 131 | – |
| 6 | 1.89 | – | 267 |

In Table 2, we show the time for DTD generation for each approach. Our approach generates DTDs in a few seconds and shows the best performance. On the contrary, XTRACT is at least 20 times slower than our approach and DDbE is about 200 times slower. As shown above, our approach achieves remarkably better performance than other methods. Furthermore, it generates the most accurate results.

## 5. Conclusion

Despite the importance of the schema, many XML documents do not have accompanying schema. Thus, many approaches for automatic schema extraction have been proposed. However, the quality of schema inferred by some approaches is poor and some approaches consume too much time to get the result. In this paper, we describe an efficient extraction of concise and accurate schema. To achieve the conciseness, we devise a restricted element content model. And to achieve the efficiency, we apply some heuristic rules, called *Relaxed Transformation Rules* and *Folding Rules*. The result of our experiment with real-life DTDs shows that our approach achieves high accuracy and is 20 to 200 times faster than existing approaches.

## Acknowledgements

We would like to thank Professor Kazuo Iwama, editor, for his help and the anonymous referees for their valuable comments. This work was supported by the Brain Korea 21 Project.

## References

[1] D. Angluin, Equivence queries and approximate fingerprints, in: Proceedings of the Workshop on Computational Learning Theory, 1989.
[2] L. Berman, A. Diaz, Data Descriptors by Example (DDbE), IBM alphaworks, http://www.alphaworks.ibm.com /tech/DDbE, 2001.
[3] T. Bray, C. Frankston, A. Malhatro, Document Content Description for XML, W3C submission, http://www.w3.org/TR/ NOTE-dcd, 1998.
[4] T. Bray, J. Paoli, C.M. Sperberg-McQueen, Extensible Markup Language (XML) 1.0, W3C Recommendation, http://www.w3.org/TR/REC-xml, 1998.
[5] A. Brazma, Efficient identification of regular expressions from representative examples, in: Proceedings of ACM COLT, 1993.
[6] D.C. Fallside, XML Schema. Part 0, W3C recommendation, http://www.w3.org/TR/xmlschema-0, 2001.
[7] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, K. Shim, XTRACT: A system for extracting document type descriptors from XML documents, in: Proceeding of ACM SIGMOD, 2000.
[8] C.H. Moh, E.P. Lim, W.K. Ng, DTD-miner: A tool for mining DTD from XML documents, in: Proceeding of International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS), 2000.
[9] Robin Cover. The XML cover pages http://www.oasis-open. org/cover/xml.html, 2001.