

On the Expressive Power and Closure Properties of XML Schema Languages

Dongwon Lee

UCLA / CSD

dongwon@cs.ucla.edu

Murali Mani

UCLA / CSD

mani@cs.ucla.edu

Makoto Murata

IBM Tokyo Research Lab.

mmurata@trl.ibm.co.jp

1 Introduction

In this paper, we present a mathematical framework using formal language theory to describe and compare multiple XML schema languages (e.g., DTD [1], XML-Schema [13], RELAX [11]). Our framework focuses on the “element trees” defined by XML schema languages and provides a good opportunity to study closure properties and expressive power of these schema languages. Especially, in this paper, we discuss about the closure properties under boolean set operations (i.e., union, intersection, difference). Such closure properties play an important role in XML data integration systems that extensively use a union or intersection operator to create a canonical view of underlying data sources. In addition, evaluating a query that returns answers from multiple XML documents (or even from a single XML document) may require to compute union as well. Knowing exactly which schema language is closed under which operations helps an application developer to choose the right XML schema language for the given application requirements.

Related Work: There have been about a dozen XML schema languages proposed lately [6], but no mathematical comparison among those languages is available. Our framework relies largely on work in the area of “regular tree languages” [2]. We introduce several subclasses of regular tree languages by imposing some restrictions. Regular hedge languages [12, 10] are also related to regular tree languages. Local tree languages and grammars have also been studied in the past [12, 8, 9].

Roadmap: The remainder of this paper is organized as follows. In Section 2, we introduce regular tree languages and regular tree grammars. In Section 3, we propose subclasses of regular tree languages and classify a few representative XML schema languages into corresponding subclasses. In Section 4, we describe the expressiveness among the proposed subclasses and study the closure properties under boolean operations. We show the application of such properties in Section 5 and finally finish with concluding remarks in Section 6.

2 Regular Tree Languages and Grammars

Regular (string) expression (regular expression in short) over alphabet Σ is defined in [3]. We typically use G to denote a grammar and $L(G)$ to denote the language that G generates.

Regular (string) grammars or even context-free grammars [3] are not suitable to describe permissible element content in DTD and other XML schema languages since they are originally designed to describe permissible *strings*, not *element trees* [8]. Instead these element trees form regular tree languages. We borrow some definitions from [2]. Context-free tree grammars also have been studied in the past [2], but we restrict ourselves to regular tree grammars.

Definition 1. (Regular Tree Grammar) [2] A regular tree grammar (RTG) is denoted by a 4-tuple $G = (N, T, S, P)$ where N is the set of non-terminal symbols, T is the set of terminal symbols, S is the set of start symbols, where $S \subseteq N$, and P is the set of production rules of the form “ $X \rightarrow a RE$ ”, where $X \in N$, $a \in T$, and RE is a regular expression over N . □

For a production rule such as $X \rightarrow a RE$, we call a the “root symbol” of the rule, and RE the “content model” of the rule. Table 1 illustrates an exemplar XML schema `book.dtd` and its regular tree grammar representation.

Lemma 1. *A regular tree grammar can be written such that there is only one rule for every non-terminal symbol $C \in N$.*

```

<!DOCTYPE book [
  <!ELEMENT book (author+, publisher)>
  <!ELEMENT author (#PCDATA)>
  <!ELEMENT publisher (EMPTY)>
  <!ATTLIST publisher Name CDATA #IMPLIED>
]>

```

(a) book.dtd

N	$=$	$\{Book, Author, Publisher, PCDATA\}$
T	$=$	$\{book, author, publisher, pcd\}$
S	$=$	$\{Book\}$
P	$Book$	$\rightarrow book(Author^+, Publisher)$
	$Author$	$\rightarrow author(PCdata)$
	$Publisher$	$\rightarrow publisher(\epsilon)$
	$PCdata$	$\rightarrow pcd\{\epsilon\}$

(b) regular tree grammar $G_{book} = (N, T, S, P)$

Table 1: A DTD and its corresponding regular tree grammar representation.

Proof. There are two cases to prove: (1) The regular tree grammar has two rules of the form $A \rightarrow a X$ and $A \rightarrow b Y$. Then, replace the latter rule with $A1 \rightarrow b Y$, and replace every occurrence of A on the RHS of a rule by $(A + A1)$. (2) The regular tree grammar has two rules of the form $A \rightarrow a X$ and $A \rightarrow a Y$. Then, replace the two rules with a single rule $A \rightarrow a (X + Y)$. This is justified since X and Y are string-regular expression by definition and the string-regular expressions are closed under union (i.e., $+$). \square

3 Subclasses of Regular Tree Languages

In this section, we introduce two subclasses of regular tree languages, which are closely related with schema languages for XML. There are other interesting subclasses, but we omit them for the lack of space. Interested readers are referred to [7].

First, we define competition of non-terminals. Subclasses of regular tree languages are later defined by imposing restrictions on such competition.

Definition 2. (Competing Non-Terminals) Two different non-terminals A and B ($A, B \in N$, $A \neq B$) are said to be *competing* with each other if (1) A and B are in the LHS of two different production rules, and (2) these production rules share the same terminal symbol in the RHS. \square

The first subclass, called *local tree grammar*, roughly corresponds to DTD; it prohibits competition of non-terminals.

Definition 3. (Local Tree Grammar and Language) A *local tree grammar (LTG)* is a regular tree grammar that does not have competing non-terminals. A language is a *local tree language* if it is generated by a local tree grammar. \square

Example 1: The following grammar $G_1 = (N, T, S, P)$ is a local tree grammar since there are no competing non-terminals.

N	$=$	$\{Book, Author, Son, PCDATA\}$
T	$=$	$\{book, author, son, pcd\}$
S	$=$	$\{Book\}$,
P	$=$	$\{Book \rightarrow book(Author), Author \rightarrow author(Son), Son \rightarrow son(PCdata), PCdata \rightarrow pcd\{\epsilon\}\}$

Observe that local tree grammars and extended context-free (string) grammars (ECFG) look similar. However, the former describes sets of trees, while the latter describes sets of strings. The parse tree set of an ECFG is a local tree language.

Local tree grammars are sometimes too restrictive. Next, we introduce a less restricted class by prohibiting competition of non-terminals within a single content model. This class roughly corresponds to XML-Schema.

Definition 4. (Single-Type Tree Grammar and Language) A regular tree grammar is said to be a *single-type tree grammar (STTG)* if (1) for each production rule, different non-terminals occurring in its content model do not compete with each other, and (2) start symbols do not compete with each other. A language is a *single-type tree language* if it is generated by a single-type tree grammar. \square

Example 2: Consider a regular tree grammar $G_2 = (N, T, S, P)$, where

$$\begin{aligned}
N &= \{\text{Book, Author1, Son, Article, Author2, Daughter}\} \\
T &= \{\mathbf{book, author, son, daughter}\} \\
S &= \{\text{Book, Article}\} \\
P &= \{\text{Book} \rightarrow \mathbf{book} (\text{Author1}), \text{Author1} \rightarrow \mathbf{author} (\text{Son}), \text{Son} \rightarrow \mathbf{son} (\epsilon), \\
&\quad \text{Article} \rightarrow \mathbf{article} (\text{Author2}), \text{Author2} \rightarrow \mathbf{author} (\text{Daughter}), \text{Daughter} \rightarrow \mathbf{daughter} (\epsilon)\}.
\end{aligned}$$

G_2 is a single-type tree grammar since no production rule has competing non-terminals in its content model. However, G_2 is not a local tree grammar since Author1 and Author2 compete with each other.

Example 3: Consider a regular tree grammar $G_3 = (N, T, S, P)$, where

$$\begin{aligned}
N &= \{\text{Doc, Para1, Para2}\} \\
T &= \{\mathbf{doc, para}\} \\
S &= \{\text{Doc}\} \\
P &= \{\text{Doc} \rightarrow \mathbf{doc} (\text{Para1, Para2}^*), \text{Para1} \rightarrow \mathbf{para} (\epsilon), \text{Para2} \rightarrow \mathbf{para} (\epsilon)\}
\end{aligned}$$

G_3 is not a single-type tree grammar. Observe that non-terminals *Para1* and *Para2* compete with each other and they occur together in the content model of the production rule for *Doc*.

Let us consider the following representative XML schema language proposals: DTD, XML-Schema, XDuce, and RELAX. All these proposals provide tree grammars.

For describing the XML schema languages, we represent a regular tree grammar as a 6-tuple $G = (N_1, N_2, T, S, P_1, P_2)$. Here N_1 represents non-terminals that produce trees, their production rules are described in P_1 , these rules are of the form $A \rightarrow a X$, where $A \in N_1, a \in T, X \in N_2$. N_2 represents non-terminals that produce a list of trees, their production rules are described in P_2 , these rules are of the form $X \rightarrow RE$, where RE is a regular expression over $N_1 \cup N_2$. T is the set of terminal symbols or labels, and $S \subseteq N_1$ is the set of start symbols.¹

DTD as defined in [1] provides local tree grammars. In DTD, there cannot be any competing non-terminals since terminal symbols and non-terminal symbols are not distinguished.

XML-Schema [13] provides single-type tree grammars. For instance, in XML-Schema, a construct `<xsd:complexType name="foo">` typically represents a production rule in P_2 :

```

<xsd:complexType name="Book">
  <xsd:sequence>
    <xsd:element name="title" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="author" minOccurs="1" maxOccurs="unbounded"/>
    <xsd:element name="publisher" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

```

This can be converted into a grammar rule $\text{Book} \rightarrow (\text{title}, \text{author}^+, \text{publisher}?)$, where $\text{title}, \text{author}, \text{publisher} \in N_1$, and $\text{Book} \in N_2$.

XDuce [4] provides type definitions equivalent to a regular tree grammar. A type definition that produces a tree is converted into a rule in P_1 . Consider the example from [4]: *type* Addrbook = addrbook[Person*] would be written as $P_1 : \text{Addrbook} \rightarrow \text{addrbook PERSON}, P_2 : \text{PERSON} \rightarrow (\text{Person}^*)$. Any type definition that does not produce a tree is written as P_2 rules. For example, *type* $X = T, X | ()$ represents the P_2 rules $X \rightarrow (T, X + \epsilon)$. Note that XDuce writes the above type rules in a right-linear form, which makes every content model definition equivalent to a regular string language.

Any regular tree grammar can be expressed in RELAX [5] via `elementRule` and `hedgeRule` [11, 5]. For instance, an `elementRule` defines a rule in P_1 and a rule in P_2 as follows [11].

```

<elementRule role="section" label="Section">
  <ref label="paraWithFNotes" occurs="*" />
</elementRule>

```

This can be converted into the production rules $P_1 : \text{Section} \rightarrow \text{section SECTION}$ and $P_2 : \text{SECTION} \rightarrow (\text{paraWithFNotes}^*)$. In addition, a `hedgeRule` defines a rule in P_2 . For instance,

¹The constraints required for this representation and its equivalence to a regular tree grammar as in Definition 1 is given in [7].

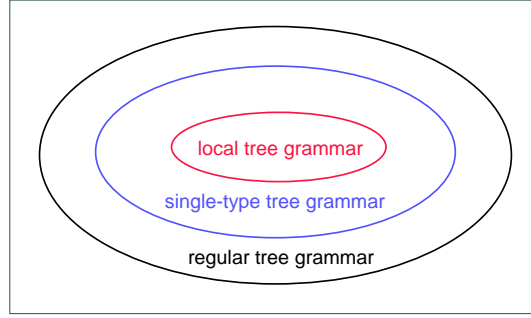


Figure 1: The expressive power of the different grammars: (a) local tree grammars (e.g., DTD), (b) single-type tree grammars (e.g., XML-Schema), and (c) regular tree grammars (e.g., RELAX, XDuce).

```
<hedgeRule label="blockElem"> <ref label="para"/> </hedgeRule>
```

This `hedgeRule` can be converted into grammar as $P2 : \text{blockElem} \rightarrow (\text{para})$. We omit the details here due to space constraint. Refer to [7] for further description.

4 Expressive Power and Closure Properties

In this section, we examine the relationship between the expressive power of the various grammar classes we introduced earlier. First, we state the following without proofs. Refer to [7] for the proofs of these relationships.

- Single-type tree grammars are strictly more expressive than local tree grammars.
- Regular tree grammars are strictly more expressive than single-type constraint grammars.

Figure 1 compares the expressive power of the different grammar classes and XML schema language proposals. Regular tree languages are closed under union, intersection and difference [2]. Let us consider other classes of tree languages.

Theorem 1. *The class of single-type tree languages and that of local tree languages are not closed under union.*

Proof. We consider two local tree grammars G_1 and G_2 , and show $L(G_1) \cup L(G_2)$ cannot be generated by a single-type tree grammar by contradiction. Let²

$$\begin{aligned} G_1 &= (\{\text{Doc}, \text{Sec1}, \text{Para}\}, \{\mathbf{doc}, \mathbf{sec}, \mathbf{para}\}, \{\text{Doc}\}, \\ &\quad \{\text{Doc} \rightarrow \mathbf{doc} (\text{Sec1}^*), \text{Sec1} \rightarrow \mathbf{sec} (\text{Para}), \text{Para} \rightarrow \mathbf{para} (\epsilon)\}), \\ G_2 &= (\{\text{Doc}, \text{Sec2}, \text{Para}\}, \{\mathbf{doc}, \mathbf{sec}, \mathbf{para}\}, \{\text{Doc}\}, \\ &\quad \{\text{Doc} \rightarrow \mathbf{doc} (\text{Sec2}^*), \text{Sec2} \rightarrow \mathbf{sec} (\text{Para}, \text{Para}^+), \text{Para} \rightarrow \mathbf{para} (\epsilon)\}). \end{aligned}$$

Obviously, G_1 generates $\langle \mathbf{doc} \rangle \langle \mathbf{sec} \rangle \langle \mathbf{para} \rangle \langle \mathbf{para} \rangle \langle \mathbf{sec} \rangle \langle \mathbf{para} \rangle \langle \mathbf{para} \rangle \langle \mathbf{sec} \rangle \langle \mathbf{doc} \rangle$. Meanwhile, G_2 generates $\langle \mathbf{doc} \rangle \langle \mathbf{sec} \rangle \langle \mathbf{para} \rangle \langle \mathbf{para} \rangle \langle \mathbf{sec} \rangle \langle \mathbf{sec} \rangle \langle \mathbf{para} \rangle \langle \mathbf{para} \rangle \langle \mathbf{sec} \rangle \langle \mathbf{doc} \rangle$. Suppose that a single-type tree grammar G_3 captures $L(G_1) \cup L(G_2)$. Then, the above trees are generated by G_3 . The derivations of these two trees begin with the same start symbol, say s ; if they are different, they compete with each other and we have a contradiction. By Lemma 1, G_3 has at most one production rule such that the LHS is s and \mathbf{doc} occurs in the RHS. Let this production rule be $s \rightarrow \mathbf{doc} e$.

Let n_{11} and n_{12} be the non-terminals from which the first and second sections in the first tree are derived, respectively. Likewise, let n_{21} and n_{22} be the non-terminals from which the first and second sections in the second tree are derived, respectively. Then, $n_{11} n_{12}$ and $n_{21} n_{22}$ are permitted by e . By the definition of single-type tree grammars, non-terminals n_{11} , n_{12} , n_{21} , and n_{22} are identical.

Now, consider a tree $\langle \mathbf{doc} \rangle \langle \mathbf{sec} \rangle \langle \mathbf{para} \rangle \langle \mathbf{para} \rangle \langle \mathbf{sec} \rangle \langle \mathbf{sec} \rangle \langle \mathbf{para} \rangle \langle \mathbf{para} \rangle \langle \mathbf{sec} \rangle \langle \mathbf{doc} \rangle$. It is easy to show that this tree is also generated by G_3 . But we have a contradiction since neither G_1 nor G_2 generates this document. \square

Observe that the union of $L(G_1)$ and $L(G_2)$ can be captured by a regular tree grammar G_3 defined below:

$$\begin{aligned} G_3 &= (\{\text{Doc}, \text{Sec1}, \text{Sec2}, \text{Para}\}, \{\mathbf{doc}, \mathbf{sec}, \mathbf{para}\}, \{\text{Doc}\}, \\ &\quad \{\text{Doc} \rightarrow \mathbf{doc} (\text{Sec1}^* + \text{Sec2}^*), \text{Sec1} \rightarrow \mathbf{sec} (\text{Para}), \text{Sec2} \rightarrow \mathbf{sec} (\text{Para}, \text{Para}^+), \text{Para} \rightarrow \mathbf{para} (\epsilon)\}). \end{aligned}$$

²For convenience, we use A^+ to denote A, A^* .

Language	Grammar class	Boolean operation		
		union	difference	intersection
DTD	local tree grammar	No	No	Yes
XML-Schema	single-type tree grammar	No	No	Yes
XDUCE	regular tree grammar	Yes	Yes	Yes
RELAX	regular tree grammar	Yes	Yes	Yes

Table 2: Summary of closure properties. “Yes” or “No” means the operation is closed or not closed, respectively.

But G_3 is not a single-type tree grammar, since non-terminals Sec1 and Sec2 compete with each other and occur in the content model of the first production rule.

Theorem 2. *The class of single-type tree languages and that of local tree languages are not closed under difference.*

Proof. We consider two grammars G_1 and G_2 , and show that $L(G_1) - L(G_2)$ cannot be generated by a single-type tree grammar by contradiction.

Let

$$\begin{aligned}
G_1 &= (\{\text{Doc}, \text{Sec1}, \text{Para}\}, \{\mathbf{doc}, \mathbf{sec}, \mathbf{para}\}, \{\text{Doc}\}, \\
&\quad \{\text{Doc} \rightarrow \mathbf{doc} (\text{Sec1}, \text{Sec1}), \text{Sec1} \rightarrow \mathbf{sec} (\text{Para}^*), \text{Para} \rightarrow \mathbf{para} (\epsilon)\}), \\
G_2 &= (\{\text{Doc}, \text{Sec2}, \text{Para}\}, \{\mathbf{doc}, \mathbf{sec}, \mathbf{para}\}, \{\text{Doc}\}, \\
&\quad \{\text{Doc} \rightarrow \mathbf{doc} (\text{Sec2}, \text{Sec2}), \text{Sec2} \rightarrow \mathbf{sec} (\text{Para}^+), \text{Para} \rightarrow \mathbf{para} (\epsilon)\}).
\end{aligned}$$

Then, G_1 generates $\langle \text{doc} \rangle \langle \text{sec} \rangle \langle \text{para} \rangle \langle / \text{sec} \rangle \langle / \text{doc} \rangle$ and $\langle \text{doc} \rangle \langle \text{sec} \rangle \langle \text{sec} \rangle \langle \text{para} \rangle \langle / \text{sec} \rangle \langle / \text{doc} \rangle$. On the other hand, G_2 generates neither of them. Thus, these trees are contained in $L(G_1) - L(G_2)$.

Assume that a single-type tree grammar G_3 captures $L(G_1) - L(G_2)$. As in the previous proof, we can show that (1) the derivations of the two trees by G_3 begin with the same start symbol, and that (2) the four sections in the above trees are derived from the the same non-terminal of G_3 .

Finally, consider a tree $\langle \text{doc} \rangle \langle \text{sec} \rangle \langle \text{para} \rangle \langle / \text{sec} \rangle \langle \text{sec} \rangle \langle \text{para} \rangle \langle / \text{sec} \rangle \langle / \text{doc} \rangle$. It is easy to show that this tree is also generated by G_3 . But we have a contradiction since G_2 generates this tree. □

Observe that $L(G_1) - L(G_2)$ can be captured by a regular tree grammar G_3 defined below:

$$\begin{aligned}
G_3 &= (\{\text{Doc}, \text{Sec1}, \text{Sec2}, \text{Para}\}, \{\mathbf{doc}, \mathbf{sec}, \mathbf{para}\}, \{\text{Doc}\}, \\
&\quad \{\text{Doc} \rightarrow \mathbf{doc} ((\text{Sec1}, \text{Sec2}) + (\text{Sec2}, \text{Sec1})), \text{Sec1} \rightarrow \mathbf{sec} (\text{Para}^*), \text{Sec2} \rightarrow \mathbf{sec} (\epsilon), \text{Para} \rightarrow \mathbf{para} (\epsilon)\}).
\end{aligned}$$

But G_3 is not a single-type tree grammar, since non-terminals Sec1 and Sec2 compete with each other and occur in the content model of the first production rule.

Theorem 3. *The class of single-type tree languages and that of local tree languages are closed under intersection.*

We merely give a hint (refer to [7] for a whole proof). The intersection of $L(G_1)$ and $L(G_2)$ can be captured by G_3 such that (1) each non-terminal of G_3 is a pair of a non-terminal of G_1 and another of G_2 , (2) each production rule of G_3 simulates a production rule of G_1 and another of G_2 , and (3) each start symbol of G_3 is a pair of a start symbol of G_1 and another of G_2 . We only have to show that thus constructed G_3 is a local tree grammar or single-type tree grammar.

The summary of the closure properties under three boolean operations for different XML schema languages are shown in Table 2.

5 Application

Expressive power and closure properties of XML schema languages have important applicability in database systems. Consider an XML-based mediation system (e.g., MIX [?]). Compared to the mediation system for semi-structured model (e.g., TSIMMIS [?]), an XML-based mediation system can take advantage of the schema-provided structure in formulating and executing queries. In such a mediation system, developers typically define a specific *view* using a view definition language (e.g., XMAS [?], MSL [?]) for mediator/wrapper for various reasons and users are allowed

```

<!DOCTYPE dept [
  <!ELEMENT dept (prof*, student*)>
  <!ELEMENT prof (name, pub*, course*)>
  <!ELEMENT student (name, pub*)>
  <!ELEMENT pub (journal|conf)>
]>

```

(a) dept.dtd

```

<!DOCTYPE view [
  <!ELEMENT view (student*)>
  <!ELEMENT student (name, pub, pub+)>
  <!ELEMENT pub (journal|conf)>
]>

```

(b) view.dtd

Table 3: DTDs for department (i.e., dept.dtd) and students with at least 2 journal papers (i.e., view.dtd). Note that using DTD notation, it is not possible to precisely express the students with at least 2 journal papers. Since view.dtd merely expresses students with at least 2 publications (either journal or conference), it could allow, for instance, students with 2 conference papers without any journal papers.

to ask queries based on such a view definition. Furthermore, to help users to formulate XML queries, XML schema-based GUIs have been developed (e.g., BBQ [?]). Therefore, the needs arise to automatically infer XML schema from the view definition of mediator/wrapper and to feed such schema into the query GUI.

Consider a situation when a view designer wants to define a view “students with at least 2 journal publications” (modified from [?]) against the schema dept.dtd in Table 3. If the designer decided to use DTD as the XML schema language for mediator system, he/she is not able to express the view precisely. The best he/she can do is the view.dtd in Table 3. The reason is that such constraint is beyond the capability of XML schema language belonging to local tree grammar (e.g., DTD). Therefore, the designer has to use more expressive XML schema languages to specify such a view in the mediation system. For instance, the following production rules P of a regular tree grammar G exactly express the specified view constraint and both XDuce and RELAX can express the grammar G easily:

$$\begin{aligned}
 P = \{ & \text{View} \rightarrow \mathbf{view} (\text{Student}^*), \text{Student} \rightarrow \mathbf{student} (\text{Name}, \text{Pub1}^*, \text{Pub2}, \text{Pub1}^*, \text{Pub2}, \text{Pub1}^*), \\
 & \text{Pub1} \rightarrow \mathbf{pub} (\text{Journal} + \text{Conf}), \text{Pub2} \rightarrow \mathbf{pub} (\text{Journal}) \}
 \end{aligned}$$

6 Conclusion

A mathematical framework using formal language theory to compare various XML schema languages is presented. Using the proposed framework, it is now possible to compare the expressiveness of content models between two XML schema languages in a precise manner. For instance, among the schema languages DTD, XML-Schema, XDuce, and RELAX, we have found that the expressive power of DTD is weaker than that of XML-Schema, which is in turn weaker than that of XDuce or RELAX. We have also found that the class of the languages DTD and XML-Schema is only closed under intersection operation while the class of the languages XDuce and RELAX is closed under union, difference, and intersection operations.

References

- [1] T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds.). “Extensible Markup Language (XML) 1.0”. W3C, Feb. 1998. <http://www.w3.org/TR/REC-xml>.
- [2] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. “Tree Automata Techniques and Applications”, 1997. <http://www.grappa.univ-lille3.fr/tata>.
- [3] J. E. Hopcroft and J. D. Ullman. “Introduction to Automata Theory, Language, and Computation”. Addison-Wesley, 1979.
- [4] H. Hosoya and B. C. Pierce. “XDuce: A Typed XML Processing Language”. In *Int’l Workshop on the Web and Databases (WebDB)*, Dallas, TX, May. 2000.
- [5] ISO/IEC. *Information Technology – Text and Office Systems – Regular Language Description for XML (RELAX) – Part 1: RELAX Core*, 2000. DIS 22250-1.
- [6] D. Lee and W. W. Chu. “Comparative Analysis of Six XML Schema Languages”. *ACM SIGMOD Record*, 29(3):76–87, Sep. 2000.
- [7] D. Lee, M. Mani, and M. Murata. “Reasoning about XML Schema Languages using Formal Language Theory”. Technical Report, IBM Almaden Research Center, RJ# 10197, Log# 95071, Nov. 2000. <http://www.cs.ucla.edu/~dongwon/paper/>.
- [8] M. Murata. “Regularity and Locality of String Languages and Tree Languages”, Feb. 1999. <http://www.oasis-open.org/cover/murataRegularity.html>.
- [9] M. Murata. “Syntax for regular-but-non-local schemata for structured documents”, Apr. 1999. <http://www.oasis-open.org/cover/murataSyntax19990311.html>.
- [10] M. Murata. “Hedge Automata: a Formal Model for XML Schemata”, 2000. http://www.xml.gr.jp/relax/hedge_nice.html.
- [11] M. Murata. “RELAX (REGular LANGUAGE description for XML)”, 2000. <http://www.xml.gr.jp/relax/>.
- [12] M. Takahashi. “Generalizations of Regular Sets and Their Application to a Study of Context-Free Languages”. *Information and Control*, 27(1):1–36, Jan. 1975.

[13] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn (Eds.). “XML Schema Part 1: Struc-

tures”, Sep. 2000. <http://www.w3.org/TR/xmlschema-1/>.