On the Expressive Power and Closure Properties of XML Schema Languages

Dongwon Lee        Murali Mani        Makoto Murata
UCLA / CSD         UCLA / CSD         IBM Tokyo Research Lab.
dongwon@cs.ucla.edu mani@cs.ucla.edu 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 \( C \in N \).
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 \( AI \rightarrow b Y \), and replace every occurrence of \( A \) on the RHS of a rule by \( (A + AI) \). (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 refered to [7].

First, we define competion 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.

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.

\[
N = \{ \text{Book, Author, Son, Pdata} \} \\
T = \{ \text{book, author, son, pdata} \} \\
S = \{ \text{Book} \}, \\
P = \{ \text{Book} \rightarrow \text{book (Author), Author} \rightarrow \text{author (Son), Son} \rightarrow \text{son (Pdata), Pdata} \rightarrow \text{pdata (e)} \} 
\]

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

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

$G_2$ 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

$$
N = \{\text{Doc}, \text{Para1}, \text{Para2}\} \\
T = \{\text{doc}, \text{para}\} \\
S = \{\text{Doc}\} \\
P = \{\text{Doc} \rightarrow \text{doc} \ (\text{Para1}, \text{Para2}), \text{Para1} \rightarrow \text{para} \ (\epsilon), \text{Para2} \rightarrow \text{para} \ (\epsilon)\}
$$

$G_3$ 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.\footnote{The constraints required for this representation and its equivalence to a regular tree grammar as in Definition 1 is given in [7].}

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 $\langle \text{xsd:complexType name="foo"} \rangle$ typically represents a production rule in $P_2$:

$$
\begin{align*}
\langle \text{xsd:complexType name="Book"} \rangle \\
\langle \text{xsd:sequence} \rangle \\
\langle \text{xsd:element name="title" minOccurs="1" maxOccurs="1"} \rangle \\
\langle \text{xsd:element name="author" minOccurs="1" maxOccurs="unbounded"} \rangle \\
\langle \text{xsd:element name="publisher" minOccurs="0" maxOccurs="1"} \rangle \\
\langle /\text{xsd:sequence} \rangle \\
\langle /\text{xsd:complexType} \rangle
\end{align*}
$$

This can be converted into a grammar rule Book $\rightarrow$ (title, author*, publisher?), where title, author, publisher $\in N_1$, and 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 $P_1$ : Addrbook $\rightarrow$ addrbook PERSON, $P_2$ : PERSON $\rightarrow$ (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.


$$
\begin{align*}
\langle \text{elementRule role="section" label="Section"} \rangle \\
\langle \text{ref label="paraWithFNotes" occurs="*"} \rangle \\
\langle /\text{elementRule} \rangle
\end{align*}
$$

This can be converted into the production rules $P_1$ : Section $\rightarrow$ section SECTION and $P_2$ : SECTION $\rightarrow$ (paraWithFNotes*). In addition, a hedgeRule defines a rule in $P_2$. For instance,
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. Let$^2$

$$G_1 = (\{\text{Doc}, \text{Sec1}, \text{Para}\}, \{\text{doc, sec, para}\}, \{\text{Doc}\},$$

$$\{\text{Doc} \rightarrow \text{doc (Sec1*)}, \text{Sec1} \rightarrow \text{sec (Para)}, \text{Para} \rightarrow \text{para (e)}\}),$$

$$G_2 = (\{\text{Doc}, \text{Sec2}, \text{Para}\}, \{\text{doc, sec, para}\}, \{\text{Doc}\},$$

$$\{\text{Doc} \rightarrow \text{doc (Sec2*)}, \text{Sec2} \rightarrow \text{sec (Para, Para*)}, \text{Para} \rightarrow \text{para (e)}\}).$$

Obviously, $G_1$ generates $<doc><sec><para/></sec><sec><para/></sec></doc>$. Meanwhile, $G_2$ generates $<doc><sec><para/>><para/>><sec><para/>><para/>></sec></doc>$. 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 doc occurs in the RHS. Let this production rule be $s \rightarrow \text{doc } e$.

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 $n_{11}, n_{12}, n_{21}, n_{22}$ are identical.

Now, consider a tree $<doc><sec><para/>><para/>><sec><para/>></sec></doc>$. 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. \[QED\]

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

$$G_3 = (\{\text{Doc, Sec1, Sec2, Para}\}, \{\text{doc, sec, para}\}, \{\text{Doc}\},$$

$$\{\text{Doc} \rightarrow \text{doc (Sec1* + Sec2*)}, \text{Sec1} \rightarrow \text{sec (Para)}, \text{Sec2} \rightarrow \text{sec (Para, Para*)}, \text{Para} \rightarrow \text{para (e)}\}).$$

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

---

**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).

This `hedgeRule` can be converted into grammar as $P2 : \text{blockEle}m \rightarrow \text{(para)}$. We omit the details here due to space constraint. Refer to [7] for further description.
Table 2: Summary of closure properties. “Yes” or “No” means the operation is closed or not closed, respectively.

<table>
<thead>
<tr>
<th>Language</th>
<th>Grammar class</th>
<th>Boolean operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTD</td>
<td>local tree grammar</td>
<td>union difference intersection</td>
</tr>
<tr>
<td>XML-Schema</td>
<td>single-type tree grammar</td>
<td>No No Yes</td>
</tr>
<tr>
<td>Xduce</td>
<td>regular tree grammar</td>
<td>Yes Yes Yes</td>
</tr>
<tr>
<td>RELAX</td>
<td>regular tree grammar</td>
<td>Yes Yes Yes</td>
</tr>
</tbody>
</table>

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

\[
G_1 = \{\text{Doc, Sec1, Para}\}, \{\text{doc, sec, para}\}, \{\text{Doc}\}, \{\text{Doc} \rightarrow \text{doc (Sec1, Sec1)}, \text{Sec1} \rightarrow \text{sec (Para*)}, \text{Para} \rightarrow \text{para (e)}\},
\]

\[
G_2 = \{\text{Doc, Sec2, Para}\}, \{\text{doc, sec, para}\}, \{\text{Doc}\}, \{\text{Doc} \rightarrow \text{doc (Sec2, Sec2)}, \text{Sec2} \rightarrow \text{sec (Para*)}, \text{Para} \rightarrow \text{para (e)}\}.
\]

Then, $G_1$ generates $<\text{doc}><\text{sec}<\text{para}/><\text{sec}</\text{sec} /> <\text{sec}><\text{para} /> </\text{sec} /> <\text{sec} /></\text{sec}>$. 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 $<\text{doc}><\text{sec}<\text{para} /> </\text{sec} /> <\text{sec}><\text{para} /> </\text{sec} /> <\text{sec} /> <\text{sec}]}</\text{sec}>$. 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:

\[
G_3 = \{\text{Doc, Sec1, Sec2, Para}\}, \{\text{doc, sec, para}\}, \{\text{Doc}\}, \{\text{Doc} \rightarrow \text{doc (Sec1, Sec2 + (Sec2, Sec1))}, \text{Sec1} \rightarrow \text{sec (Para*)}, \text{Sec2} \rightarrow \text{sec (e)}, \text{Para} \rightarrow \text{para (e)}\}.
\]

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
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:

$$P = \{ \text{View} \to \text{view (Student$^*$)}, \text{Student} \to \text{student (Name Pub1$^*$, Pub2, Pub1$^*$, Pub2, Pub1$^*$), Pub1} \to \text{pub (Journal + Conf), Pub2} \to \text{pub (Journal)} \}$$

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


open.org/cover/murataRegularity.html.

non-local schemata for structured docu-
ments”, Apr. 1999. http://www.oasis-
open.org/cover/murataSyntax19990311.html.

