# Summary: Automata, Logic, and XML

Topic 4: Foundations of XML - No. 1

s1856331: Muthuri Mwenda - March 2019

**Abstract**

This paper reviews the progress made in the interconnection between automata and logic in relation to XML. Emphasis is laid on unranked tree autoamata, tree-walking automata and automata over infinite alphabets and their connections with First Order and Monadic Second Order logic. It highlights how application of automata theory in XML poses a unique set of challenges in relation to this trio and concludes by offering pointers to ideal starting points for future research.

## 1 Introduction

Current technology compels modelling data that is not tabular. This has led to the adoption of Extensible Markup Language (XML) as a standard data exchange format by the World Wide Web Consortium, where data is modelled as labelled ordered attributed trees. This new data model requires new tools and techniques to evaluate its complexity. Enter tree automata theory. Recall that with relational databases, while exploiting the applicability of finite model theory, provided a concrete scenario for the theory and brought up questions unlikely to have risen independently [30]. Similarly, XML poses new challenges in automata and logic arising from its actualization. In this regard, application of automata to XML can be broadly divided into the following categories:

1. As a formal model of computation
2. As a means of evaluating query and pattern languages
3. As a formalism for describing schemas
4. As an algorithmic toolbox

The paper surveys the application of three well-studied automata formalisms necessitated by research in XML: unranked tree automata, tree-walking automata and automata over infinite alphabets. While the first two ignore attributes and text values, automata over infinite datasets incorporates these data elements. For each of the three models, their relation with XML is discussed, recent results are surveyed and new research directions are suggested.

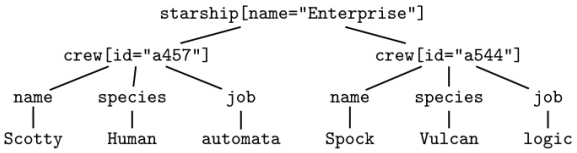
## 2 Basics of XML

XML is a simple, flexible data-exchange format whose documents can be accurately represented as labelled attributed ordered trees [24]. Consider the XML document in Figure 1 displaying some information about a spaceship and its crew members. Analogous to the Hypertext Markup Language (HTML) syntax, XML uses elements delimited by start and end tags. Elements can be arbitrarily nested inside other elements. For instance, the element <name>Spock </name>is a subelement of the outer crew-element. Elements can have attributes (name value pairs separated by an equality sign), but the value of attributes is always atomic; they cannot be nested. Attributes appear in the start-tag of the element they belong to. An example is <starship name=“Enterprise”>indicating that the value of the name attribute belonging to that particular starship-element is Enterprise.
An XML document can be viewed as a tree. The outermost element is the root and every subelement has its children. An attribute of an element is an attribute of the corresponding node. An alternative representation is encoding attributes as child nodes of the elements to which they belong. The paper remarks that there is no unique best method of representing XML documents as trees.

Usually, XML documents satisfying some specific constraints are explored. Specification of such a schema uses Document Type Definitions (DTDs). DTDs are context-free grammars with regular expressions at their right-hand side. Figure 2(a) shows a specification of the data type of a spaceship. The outermost element is the starship, every crew element has a name and species as the first and second subelements respectively and rank or job as the third subelement. | and , denote disjunction and

```
<starship name="Enterprise">
  <crew id="a457">
    <name> Scotty </name>
    <species> Human </species>
    <job> automata </job>
  </crew>
  <crew id="a544">
    <name> Spock </name>
    <species> Vulcan </species>
    <job> logic </job>
  </crew>
</starship>
```

(a) Example of an XML document




(b) Tree representation of XML document

Figure 1: An XML document and its tree representation

concatenation respectively. #PCDATA indicates that the element has no subelements but contains data only. ATTLIST determines the attribute that belongs to a certain element. Attributes specified in this DTD can only have a single string value. Attributes can be used to link nodes. As is in Figure 2(b), the id a457 of Scotty can be used in a different place of the document to refer to the node.

In XML representation by trees, inner nodes correspond to elements whereas leaf nodes contain arbitrary text. Excluding the exploration of automata over infinite alphabets in section 6, the paper focuses on the structure of XML documents and ignores attributes and text in the leaf nodes. Accordingly, XML documents are trees over a finite alphabet where the alphabet is determined by a DTD. The nodes of these trees can have an arbitrary number of children, hence referred to as unranked trees [9, 28].

```
<!DOCTYPE starship [
    <!ELEMENT starship (crew)*>
    <!ELEMENT crew     (name,species,(rank | job))>
    <!ELEMENT name     (#PCDATA)>
    <!ELEMENT species  (#PCDATA)>
    <!ELEMENT rank     (#PCDATA)>
    <!ATTLIST starship name   CDATA>
    <!ATTLIST crew     id     CDATA>
]>
```

(a) A DTD describing structure of XML document of Figure 1

```
<cabin>
  <number> 988 </number>
  <inhabitant> a457 </inhabitant>
</cabin>
```

(b) Using the id-attribute

Figure 2

# 3 Trees and Logic

## 3.1 Trees

We fix a finite alphabet $\Sigma$ of element names. The set of $\Sigma$-trees is denoted by $T_\Sigma$ and inductively defined as follows: $(i)$every $\sigma \in \Sigma$ is a $\Sigma$-tree; $(ii)$if $\sigma \in \Sigma$ and $t_1, ..., t_n \in T_\Sigma$, $n \geq 1$ then $\sigma(t_1, ..., t_n)$ is a $\Sigma$-tree. Since there is no priory bound on the number of children for a node in a $\Sigma$-tree, it is unranked. For every $t \in T_\Sigma$, the set of nodes of $t$ denoted by $\mathrm{Dom}(t)$, is the subset of natural numbers defined as: if $t = \sigma(_{t_1 ... t_n})$ with $\sigma \in \Sigma, n \geq 0$, and $t_1, ..., t_n \in T_\Sigma$, then $\mathrm{Dom}(t) = \{\varepsilon\} \cup \{iu \mid i \in \{1, ..., n\}, u \in \mathrm{Dom}(t_i)\}$. $\varepsilon$ represents the root while $vj$ represents the $j$-th child of $v$. $\mathrm{lab}^t(u)$ denotes the label of $u$ in $t$. In this paper, tree means $\Sigma$-tree.

**Definition 1.** A *DTD* is a tuple $(d, s_d)$ where $d$ is a function that maps $\Sigma$-symbols to regular expressions over $\Sigma$ and $s_d \in \Sigma$ is the start symbol. In the sequel, $d$ is used in the place of $(d, s_d)$.

A tree $t$ satisfies $d$ iff $\mathrm{lab}^t(\varepsilon) = s_d$ and for every $u \in \mathrm{Dom}(t)$ with $n$ children, $\mathrm{lab}^t(u1)...\mathrm{lab}^t(un) \in d(\mathrm{lab}^t(u))$. If $u$ has no children, $\varepsilon$ should belong to $d(\mathrm{lab}^t(u))$.

*Intuition:* From the DTD on Figure 2(a), name, species, rank and job would have $d(\mathrm{lab}^t(u))$ as $\varepsilon$ because we are only interested in the structure of the document, ignoring data elements.

## 3.2 Logic

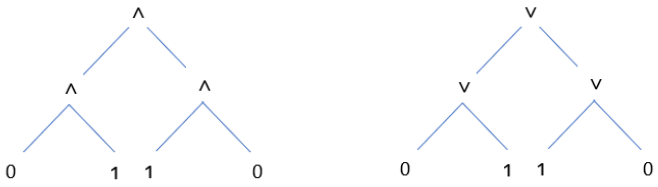
Trees can be viewed as logical structures as illustrated by Ebbinghaus [7]. The relational vocabulary $_{T\Sigma} := \{E, <, (O_\sigma)_{\sigma \in \Sigma}\}$ is used where $E$ and $<$ are binary and all the $O\sigma$ are unary relation symbols. The domain of $t$, viewed as a structure, equals the set of nodes of $t$; $\mathrm{Dom}(t)$. In addition, $E$ is the edge

relation and equals the set of pairs $(v, v \cdot i)$ where $v, v \cdot i \in \text{Dom}(t)$. The relation $<$ specifies the ordering of children of a node and equals the set of pairs $(v \cdot i, v \cdot j)$ where $i < j$ and $v \cdot j \in \text{Dom}(t)$. For each $\sigma$, $O\sigma$ is the set of nodes that are labelled with a $\sigma$.
The paper considers first-order (FO) and monadic second-order logic (MSO) over these logical structures. MSO is FO extended with quantification over set variables [7, 28].

# 4 Unranked Tree Automata [3, 22]

## 4.1 Definition

**Definition 2.** A *nondeterministic tree automaton (NTA)* is a tuple $B = (Q, \Sigma, \delta, F)$, where $Q$ is a finite set of states, $F \subseteq Q$ is the set of final states and $\delta$ is a function $Q \times \Sigma \longrightarrow 2^{Q^*}$ such that $\delta(q, a)$ is a regular sting over language over $Q^*$ for every $a \in \Sigma$ and $q \in Q$. A run of $B$ on tree $t$ is a labelling $\lambda : \text{Dom}(t) \mapsto Q$ such that for every $v \in \text{Dom}\ (t)$ with $n$ children, $\lambda(v1) \cdots \lambda(vn) \in \delta(\lambda(v),\text{lab}^t(v))$. A run is accepting iff $\lambda(\varepsilon) \in F$. A tree is accepted if there is an accepting run. The set of all accepted trees is denoted by $L(B)$. A set of trees is called *regular* when it can be recognized by an NTA.
*Example:* Given $\sum = \{\wedge, \vee, 0, 1\}$ and the trees in Figure 3, B assigns 0 and 1 to leaves and $\wedge$ and $\vee$ to non-leaf nodes. The automata accepts tree 3(b) because it reaches the final state as a 1 and rejects 3(a).




(a) Rejecting automata (b) Accepting automata

Figure 3

## 4.2 Connection with ranked trees

Recall that unranked trees can be uniformly encoded as binary trees [14]. Martens denotes that minimizing the number of states of a finite unranked tree automaton is relevant for classes of deterministic automata because the minimization can be done efficiently and yields a unique canonical representation. The first child of a node remains the first child of that node in the encoding but it is explicitly encoded as a left child. The remaining children are right descendants of the first child. If there is a right child but no left child, a # is inserted. When only the left child exists, a # is inserted for the right child. Decoding is the reverse process. From these encodings, the following proposition is obtained. Note that transition functions of NTAs are represented by nondeterministic finite automatas (NFAs).
**Proposition 1.** [27]

1. *For every unranked NTA $B$ there is a binary tree automaton $A$ such that $L(A) = \{enc(t) \mid t \in L(B)\}$. The size of $A$ is polynomial in the size of $B$.*
2. *For every binary tree automaton $A$ there is an unranked NTA $B$ such that $L(B) = \{dec(t) \mid t \in L(A)\}$. The size of $B$ is polynomial in the size of $A$.*

This proposition allows the transfer of all closure properties of ranked tree automata to the class of unranked tree automata.

## 4.3 Expressiveness and complexity

Since enc and dec are MSO definable, Proposition 1 implies that the Doner-Thatcher-Wright characterization of ranked tree automata carries over to unranked trees [6, 26].

**Corollary 1.** [19] *A set of trees $L$ is regular iff there is a MSO formula $\varphi$ such that $L = \{t \mid t \models \varphi\}$.*
Proposition 1 provides a tool for transferring results from ranked to unranked trees. However, it does not deal with issues specific to unranked tree automata. As an example, complexity decision problems for NTAs depend on the formalism used to represent regular string languages $\delta(q, a)$ in the transition function. Since there are many ways to represent regular strings, Proposition 1 seems unhelpful. Another challenge highlighted by [25] is that this *first-child-next-sibling* encoding makes it complicated to identify vertical paths of the original tree.

*Remark:* The transition between binary and unranked automata in Proposition 1 is polynomial for NTA(NFA)'s. On this account, unranked automata class can be seen as the default for unranked tree automata. For this class, the complexity of membership problem is tractable.

### 4.4 Applications

Unranked tree automata can serve XML research in the following four different ways:

1. As a **basis of schema languages** and **validating schemas**. First considered by Murata [16], tree automata has been used as a schema definition language by Relax [29]. In addition, XDuce [10] and specialized DTDs by Papakonstantinou et al. [23] correspond precisely to the unranked tree languages. Further, Lee et al. [13] compare XML schema languages based on formal language theory.
2. As an **evaluation mechanism** for pattern languages. Researchers have defined pattern languages for unranked trees that can be implemented by unranked tree automata. For instance, [17] defines an extension of path expressions based on regular expressions over unranked trees. Another application is [18].
3. As an **algorithmic toolbox**. Binary tree automata has been used to obtain an algorithm for XPath containment by Miklau and Suciu [15].
4. As a new **paradigm**. Since unranked tree automata uses regular string languages to deal with unrankedness, [21] define query automata being two-way deterministic unranked tree automata that can select nodes in the tree. Query automata corresponds exactly to the unary queries definable in MSO.

## 5 Tree-walking automata

Computation by tree walking has been well-studied from formal language theory in the context of attribute grammars and tree transformations [1, 2, 5] and has materialized in XML research in various ways. As a first, tree walking is provided by the context of caterpillar expressions of Bruggemann-Klein and Wood [4]. This section explores ordinary tree-walking automata.

### 5.1 Definition

A tree-walking automaton is a finite state device walking a *tree*. Its control is always at one node of the input tree and based on the label of that node, its state and its position in the tree, the automaton changes state and moves to one of the neighbouring nodes. The automaton accepts if it enters a final state.

**Definition 3.** A TWA $C$ is a tuple $(Q, \Sigma, q_0, q_F, P)$ where $Q$ is a finite set of states, $q_0 \in Q$ is the initial state, $q_F \in Q$ is the final state, $P$ is a finite set of rules of the form $(q, \sigma) \to (q', d)$ where $\sigma \in \Sigma$,$q, q' \in Q$, and $d \in \{\leftarrow, \uparrow, \rightarrow, \downarrow, stay\}$.

$C$ is deterministic if there is at most one rule $(q, \sigma) \to \alpha$ in $P$ for every $\sigma \in \Sigma$ and $q \in Q$. The class of deterministic TWAs is denoted by DTWA.

### 5.2 Expressiveness

Ranked TWAs are defined like unranked ones with the only difference being that they have a fixed $n$ such that only trees of rank $n$ are considered as inputs. A tree of rank $n$ constraints every node to have $n$ or fewer children. As with unranked tree automata, we can transfer results between ranked and unranked cases. Let *enc* and *dec* be encoding and decoding respectively, as explained in Section 4.2.

**Proposition 2.** *1. For every unranked (D)TWA $B$ there is a ranked (D)TWA $C$ such that $L(C) = \{enc(t) \mid t \in L(B)\}$. The size of $C$ is linear in the size of $B$.*

*2. For every ranked (D)TWA $C$ there is an unranked (D)TWA $B$ such that $L(B) = \{dec(t) \mid t \in L(C)\}$. The size of $B$ is linear in the size of $C$.*

Following results depicted by Engelfriet [8], the class of tree languages accepted by tree-walking pebble automata is included in the class of regular tree languages. From Proposition 1 and Proposition 2, unranked TWAs only define regular tree languages.

**Proposition 3.** *An unranked tree language is accepted by an alternating TWA (ATWA) iff it is regular.* TWAs can be used as an algorithmic toolbox with the upper bound on the non-circularity test of

extended attribute grammars obtained by a reduction to the emptiness test of TWAs.

*Remark on robustness*: Ranked tree automata can check the label of the parent of the current node by remembering the child number in the state, moving up and moving back down to the correct child hence can evaluate boolean circuits with a fixed fan-in. Unranked tree automata, in contrast, seem not to achieve this as the child number is unbounded.

### 5.3 A logical characterization

For every $m > 0$ the unary predicate $\text{depth}_m$ is added to the vocabulary of trees. In all trees, $\text{depth}_m$ will contain all vertices the depth of which is a multiple of $m$. Tree-walking is characterized by *transitive closure logic formulas* (TC logic) of a special form. A TC formula in normal form is an expression of the form $TC[\varphi(x,y)](\varepsilon,\varepsilon)$, where $\varphi$ is a FO formula which may make use of the predicate $\text{depth}_m$, for some $m$, in addition to $E$, $<$ and the $O_\sigma$. The *Deterministic transitive closure logic formulas* (DTC) are used in an analogously defined normal form to capture deterministic tree-walking automata.

**Theorem 1.** *1. A ranked tree language is accepted by a nondeterministic tree-walking automaton iff it is definable by a TC formula in normal form.*
*2. A ranked tree language is accepted by a deterministic tree-walking automaton iff it is definable by a DTC formula in normal form.*

Hanf's Theorem [7] is used to show that every TWA can evaluate a TC formula in normal form. Since unranked trees are unbounded, it is not clear whether this results are applicable.

*Intuition:* For bounded graphs, whether a FO sentence holds depends only in the number of pairwise disjoint spheres of each isomorphism type of some fixed radius. The result implies that any lower bound on (D)TC formulas in normal form is also a lower bound for (non)deterministic tree-walking automata.

## 6 Tree-walking and data-values

Data values attached to leaves or to attributes in some cases make the difference between decidability and undecidability. Therefore, we explore tree-walking with registers and look-ahead to extend the automata and logic formalisms to trees with data values.

### 6.1 Trees and logic revisited

When dealing with text, all text occurring at leaves is moved into attributes. Additionally, we assume an infinite domain $\mathbf{D} = \{d1, d2, ...\}$ and an infinite set of attributes $A$.

**Definition 4.** An *attributed* $\Sigma$*-tree* is a pair $(t, (\lambda^t_a)_{a\in A})$, where $t \in T_\Sigma$ and for each $a \in A$, $\lambda^t_a : \text{Dom}(t) \mapsto \mathbf{D}$ is a function defining the $a$-attribute of nodes in $t$. Further, XML documents can contain elements with mixed content such as nesting different tags. For the stated logic, an extended vocabulary is used; ${}_{T\Sigma,A} = \{E, <, \prec, (O_\sigma)_{\sigma\in\Sigma}, (\text{val}_a)_{a\in A}\}$. Each $\text{val}_a$ is a function, from $\text{Dom}(t)$ to $\mathbf{D}$. Quantification over $\mathbf{D}$ is not possible.

### 6.2 Tree-walking automata extended with registers

To deal with data values, a tree-walking automaton is equipped with a finite number of registers into which it can store data-values and can check whether an attribute value of the current node is equal to the content of some register [12].

**Definition 5.** *A k-register DTWA B is a tuple* $(Q, q_0, q_F, {}_{T0}, P)$ where $Q$ is a finite set of states, $q_0 \in Q$ is the initial state and $q_F \in Q$ is the final state, ${}_{T0}$ is the initial register assignment and $P$ is a finite set of rules of the form $(\sigma, q, \xi) \to \alpha$. $\sigma \in \Sigma, q \in Q$ *and* $\xi$ is a Boolean combination of atomic formulas of the form $j = b$ where $j \in \{1, ...k\}$ and $b \in A$. $\alpha$ can be either:

- $(q', d)$ with $q' \in Q$ and $d$ being a change to state $q'$ and moving in the specified direction $d$ or
- $(q', i, a)$ where $q' \in Q, i \in \{1, ...k\}$, and $a \in A$. i.e. an instruction to change to state $q'$ and replace the content of register $i$ by the value of attribute $a$.

This definition assumes that no transition is possible from the final state and that the automaton never moves off the input tree.

### 6.3 Expressiveness

Expressiveness of a $k$-register DTWA can compute properties not in MSO* (* denotes a difference with logics with the previous section.) but cannot compute all FO*-definable properties. This is an unexpected

outcome because in database theory, FO logic is generally accepted as the minimum expressiveness that a query language should have while MSO, due to its correspondence with various automata, stands for regularity and robustness. To simulate $k$-register DTWAs, we need a more powerful protocol where the number of messages depends on the number of different data values in input, hence the argument below.

**Theorem 2.** [20]

1. *MSO* cannot define all properties computable by k-register DTWAs*
2. *k-register DTWAs cannot compute all properties definable in FO*.*

As proof, the paper demonstrates an implication derived from communication complexity [11] that no $k$-register DTWA accepts a language $L$ definable in FO*. Additionally, the proof can be extended to show that even *alternating* $k$-register automata cannot compute all FO*-definable properties.

### 6.4 Subcomputations and relational storage

Consider DTWAs that are extended in two ways: *(i)* registers can store arbitrary relations over **D** and *(ii)* subcomputations can be started. The paper demonstrates that these additions are incapable of capturing FO*, thus extend the DTWA with look-ahead and relational storage.

**Theorem 3.** *DTWA automata extended with look-ahead and relational storage do not capture FO*.*

A proof is also derived from communication complexity [11]. The weakness of register automata is that when they leave a node, they usually cannot relocate that node. If there is a fixed attribute such that for every node the value of that attribute is unique among all nodes in a tree, the paper shows that various restrictions capture natural complexity classes, stated below:

**Theorem 4.** In the presence of unique ids, DTWAs extended with: (*i*)single-valued registers capture LOGSPACE, (*ii*)single-valued registers and subcomputations capture PTIME, (*iii*)relational storage capture PSPACE (*iv*)relational storage and subcomputations capture EXPTIME. These characterizations are obtained for a Turing Machine model directly operating on attributed trees. These proofs provide a relatively complete picture of the expressiveness of query languages based on tree-walking.

## 7 Conclusion

Not much is known about tree-walking automata and it is unclear whether and in which way the addition of pebbles to the formalisms increases expressiveness. As an algorithmic toolbox, register automata are unuseful as almost all problems are undecidable. Notwithstanding, in the context of streaming or typechecking with data values, it would be interesting to find the most expressive formalism for which emptiness would remain decidable. Further, it would be helpful to have general results on the complexity of unranked tree automata in terms of complexity of the regular languages representing the transition functions.

## References


[1] Alfred V Aho and Jeffrey D Ullman. Translations on a context free grammar. *Information and Control*, 19(5):439–475, 1971.

[2] Roderick Bloem and Joost Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *Journal of Computer and System Sciences*, 61(1):1–50, 2000.

[3] Anne Brüggemann-Klein, Makoto Murata, and Derick Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1. 2001.

[4] Anne Brüggemann-Klein and Derick Wood. Caterpillars: A context specification technique. In *Markup Languages*. Citeseer, 2000.

[5] Pierre Deransart, Martin Jourdan, and Bernard Lorho. *Attribute grammars: definitions, systems and bibliography*, volume 323. Springer Science & Business Media, 1988.

[6] John Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4(5):406–451, 1970.

[7] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Springer Science & Business Media, 2005.

[8] Joost Engelfriet and Hendrik Jan Hoogeboom. Tree-walking pebble automata. In *Jewels are forever*, pages 72–83. Springer, 1999.

[9] Ferenc Gécseg and Magnus Steinby. Tree languages. In *Handbook of formal languages*, pages 1–68. Springer, 1997.

[10] Haruo Hosoya, Jérôme Vouillon, and Benjamin C Pierce. Regular expression types for xml. In *ACM SIGPLAN Notices*, volume 35, pages 11–22. ACM, 2000.

[11] Juraj Hromkovič. *Communication complexity and parallel computing*. Springer Science & Business Media, 2013.

[12] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

[13] Dongwon Lee, Murali Mani, and Makoto Murata. Reasoning about xml schema languages using formal language theory. Technical report, Technical report, IBM Almaden Research Center, 2000. Log, 2000.

[14] Wim Martens and Joachim Niehren. Minimizing tree automata for unranked trees. In *International Workshop on Database Programming Languages*, pages 232–246. Springer, 2005.

[15] Gerome Miklau and Dan Suciu. Containment and equivalence for an xpath fragment. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 65–76. ACM, 2002.

[16] Makoto Murata. Data model for document transformation and assembly. In *International Workshop on Principles of Digital Document Processing*, pages 140–152. Springer, 1998.

[17] Makoto Murata. Extended path expressions of xml. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 126–137. ACM, 2001.

[18] Andreas Neumann and Helmut Seidl. Locating matches of tree patterns in forests. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 134–145. Springer, 1998.

[19] Frank Neven and Thomas Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1-2):633–674, 2002.

[20] Frank Neven, Thomas Schwentick, and Victor Vianu. Towards regular languages over infinite alphabets. In *International Symposium on Mathematical Foundations of Computer Science*, pages 560–572. Springer, 2001.

[21] Frank Neven and Jan Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. *Journal of the ACM (JACM)*, 49(1):56–100, 2002.

[22] Claude Pair and Alain Quéré. Définition et etude des bilangages réguliers. *Information and Control*, 13(6):565–593, 1968.

[23] Yannis Papakonstantinou and Victor Vianu. Dtd inference for views of xml data. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 35–46. ACM, 2000.

[24] Erik T Ray. *Learning XML: creating self-describing data.* " O'Reilly Media, Inc.", 2003.

[25] Thomas Schwentick. Automata for xml—a survey. *Journal of Computer and System Sciences*, 73(3):289–315, 2007.

[26] Dirk Siefkes. Jw thatcher and jb wright. generalized finite automata theory with an application to a decision problem of second-order logic. mathematical systems theory, vol. 2 (1968), pp. 57–81. *The Journal of Symbolic Logic*, 37(3):619–620, 1972.

[27] Dan Suciu. Typechecking for semistructured data. In *International workshop on database programming languages*, pages 1–20. Springer, 2001.

[28] Wolfgang Thomas. Languages, automata, and logic. In *Handbook of formal languages*, pages 389–455. Springer, 1997.

[29] Eric Van der Vlist. *Relax NG: A Simpler Schema Language for XML.* " O'Reilly Media, Inc.", 2003.

[30] Victor Vianu. Databases and finite-model theory. *Descriptive complexity and finite models*, 31:97–148, 1997.