Automata theory for XML researchers*

Frank Neven
University of Limburg
frank.neven@luc.ac.be

1 Introduction

The advent of XML initiated a symbiosis between document research, databases and formal languages (see, e.g., the survey by Vianu [38]). This symbiosis resulted, for instance, in the development of *unranked* tree automata [3]. In brief, unranked trees are finite labeled trees where nodes can have an arbitrary number of children. So, there is no fixed rank associated to each label. As the structure of XML documents can be adequately represented by unranked trees, unranked tree automata can serve XML research in four different ways: (*i*) as a basis of schema languages [10, 12, 13, 18, 19] and validating of schemas; (*ii*) as an evaluation mechanism for pattern languages [4, 32, 24]; (*iii*) as an algorithmic toolbox (e.g., XPath containment [16] and typechecking [15]); and (*iv*) as a new paradigm: unranked tree automata use regular string languages to deal with unrankedness. The latter simple but effective paradigm found application in several formalisms [14, 20, 21, 22, 27].

The present paper is an attempt to provide a gentle introduction to unranked tree automata and to give references to some applications. We mention that Vardi, already in 1989, wrote a paper demonstrating the usefulness of ranked tree automata for the static analysis of datalog programs [37].

2 Trees

Every XML document can be represented by a tree. See, for instance, Figure 1. In this view, inner nodes correspond to elements which determine the structure of the document, while the leaf nodes and the attributes provide the content. Sometimes, for instance in the case of typechecking, we are only interested in the structure of documents, not in the actual values of the attributes or the leaf nodes. In such cases, we can adequately represent XML documents as trees over a finite alphabet. Again, this is a restriction as the alphabet is only known when some schema information is available (such as a DTD for instance). For now, however, we fix a finite alphabet Σ and come back to the finite alphabet restriction in Section 5.

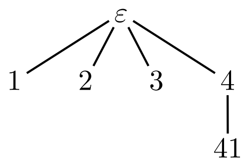
We next provide the definitions for our abstraction of XML documents: Σ-trees. The set of Σ-trees, denoted by $\mathcal{T}_\Sigma$, is inductively defined as follows:

(*i*) every $\sigma \in \Sigma$ is a Σ-tree;

(*ii*) if $\sigma \in \Sigma$ and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma$, $n \geq 1$ then $\sigma(t_1, \ldots, t_n)$ is a Σ-tree.

Note that there is no a priori bound on the number of children of a node in a Σ-tree; such trees are therefore *unranked*. Denote by $\mathbb{N}^*$, the set of strings over the alphabet consisting of the natural numbers. For every tree $t \in \mathcal{T}_\Sigma$, the *set of nodes of* t, denoted by $\mathrm{Dom}(t)$, is the subset of $\mathbb{N}^*$ defined as follows: if $t = \sigma(t_1 \cdots t_n)$ with $\sigma \in \Sigma$, $n \geq 0$, and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma$, then $\mathrm{Dom}(t) = \{\varepsilon\} \cup \{ui \mid i \in \{1, \ldots, n\}, u \in \mathrm{Dom}(t_i)\}$. Thus, ε represents the root while ui represents the i-th child of u. For instance, the domain of the bottom tree in Figure 1 is graphically represented as follows:

*Database Principles Column. Column editor: Leonid Libkin, Department of Computer Science, University of Toronto, Toronto, Ontario M5S 3H5, Canada. E-mail: libkin@cs.toronto.edu.



By $\mathrm{lab}^t(u)$ we denote the label of u in t. In the following, when we say tree we always mean Σ-tree.

3 Unranked Tree Automata

3.1 Automata on strings

Before we define tree automata, let us recall nondeterministic finite automata on strings (NFAs). In our definition we have initial states for every σ-symbol. In this way, we can define a run of the automaton that starts on the first symbol and ends on the last one. In particular, an NFA is a tuple $M = (Q, \Sigma, \delta, (I_\sigma)_{\sigma\in\Sigma}, F)$ where Q is the set of states; for every $\sigma \in \Sigma$, $I_\sigma \subseteq Q$ is the set of initial states for σ; $F \subseteq Q$ is the sets of final states; and, $\delta : Q \times \Sigma \to 2^Q$ is the transition function. A run $\lambda : \{1, \ldots, n\} \to Q$ on a string $a_1 \cdots a_n \in \Sigma^*$ is a function such that

(*i*) $\lambda(1) \in I_{a_1}$;

(*ii*) for all $i \in \{2, \ldots, n\}$, $\lambda(i) \in \delta(\lambda(i-1), a_i)$.

A run is *accepting* if $\lambda(n) \in F$. A string is accepted if there is an accepting run for it.

Example 3.1. Consider the automaton M accepting all the strings over $\{a, b\}$ that start and end with the same symbol. Set $Q := \{q_a, q_b\}$, $I_a := \{q_a\}$, $I_b := \{q_b\}$, $F := Q$, $\delta(q_a, a) := \delta(q_a, b) = \{q_a\}$, and $\delta(q_b, a) := \delta(q_b, b) = \{q_b\}$. This automaton has only one run on every string: for instance, on the string $abaa$, we have the run

	a	b	a	a
λ	q_a	q_a	q_a	q_a

□

Usually, we view NFAs as processing input strings from left to right: an initial state is assigned to the first position and from there on we assign new states that are consistent with the transition function. The string is accepted if we can assign a final state to the last position. Note, however, that we can also view M as a right-to-left automaton by simply reversing the roles of F and I. Indeed, we assign a "final" state to the last position and assign new states starting from that position to the left that are consistent with the transition function. The string is accepted if we can assign an initial state from the correct set to the first position. Although processing strings from right to left might seem a bit strange, for trees processing in a bottom-up way is equally sensible as processing in a top-down manner.

3.2 Automata on binary trees

For ease of exposition we assume for this section that all trees are binary. That is, all non-leaf nodes have exactly two children. A tree automaton is then a tuple $A = (Q, \Sigma, \delta, (I_\sigma)_{\sigma\in\Sigma}, F)$ where Q, F, and all I_σ's are as before, and $\delta : Q \times Q \times \Sigma \to Q$ is a function mapping a *pair* of states and a symbol to a new state. A run for t is a mapping $\lambda : \mathrm{Dom}(t) \to Q$ such that

(*i*) for every leaf node u, $\lambda(u) \in I_{\mathrm{lab}^t(u)}$;

(*ii*) for all inner nodes u,

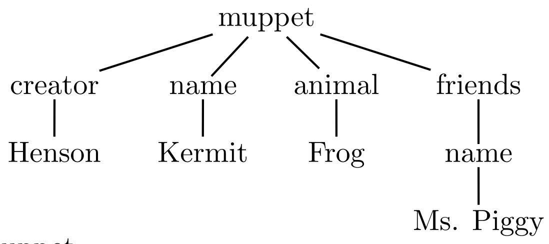
$$\lambda(u) \in \delta(\lambda(u1), \lambda(u2), \mathrm{lab}^t(u)).$$

Recall that ui is the i-th child of u.

A run is *accepting* if $\lambda(\varepsilon) \in F$. A tree is accepted if there is an accepting run for it.

Tree automata can be viewed as processing their input in a bottom-up or in a top-down fashion depending on the roles of F and I_σ. Indeed, in the bottom-up view, initial states from the I_σ's are assigned to σ-labeled leaves and new states are assigned to inner nodes depending on their label and on the states at their children. The tree is accepted if a final state is assigned to the root. In the top-down view, a 'final' state from F is assigned to the root; new states are assigned to the children of a node u depending on the label of u and the state of u. A tree is accepted if every σ-labeled leaf is assigned a state from I_σ.

```
<muppet creator="Henson">
  <name> Kermit </name>
  <animal> Frog </animal>
  <friends>
     <name> Ms. Piggy </name>
  </friends>
</muppet>
```



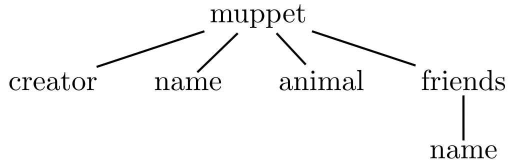


Figure 1: A sample XML document, its tree representation, and its structural representation.

Example 3.2. Consider the alphabet $\Sigma = \{\wedge, \vee, 0, 1\}$. Suppose for ease of exposition that trees are always as follows: 0 and 1 only appear at leaves, and $\wedge$ and $\vee$ can appear everywhere except at leaves. These are all tree-shaped boolean circuits. We next define an automaton accepting exactly the circuits evaluating to 1. Define $A = (Q, \Sigma, \delta, F)$ with $Q = \{0, 1\}$, $I_0 = \{0\}$, $I_1 = \{1\}$,$F = \{1\}$, and

$$\begin{array}{ll} \delta(0,0,\wedge) = 0 & \delta(0,1,\wedge) = 0; \\ \delta(1,0,\wedge) = 0 & \delta(1,1,\wedge) = 1; \\ \delta(0,0,\vee) = 0 & \delta(0,1,\vee) = 1; \\ \delta(1,0,\vee) = 1 & \delta(1,1,\vee) = 1. \end{array}$$

Intuitively, A works as follows: A assigns 0 (1) to 0-labeled (1-labeled) leaves; further, A assigns a 1 to a $\wedge$-labeled node iff both its children are 1; and, A assigns a 0 to a $\vee$-labeled node iff both its children are 0. Finally, A accepts when the root is labeled with 1. So, in this example, the bottom-up view is the most intuitive one. In Figure 2, we give an example of an accepting run on a tree.

In Example 3.4, we give an automaton evaluating an XPath expression for which the top-down view is more natural. □

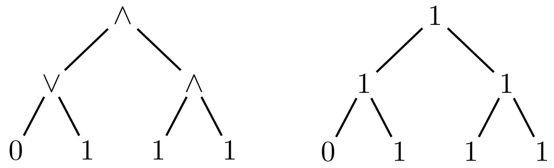


Figure 2: A tree and an accepting run of the automaton of Example 3.2.

3.3 Automata on unranked trees

Extending the tree automata of the previous section to unranked trees implies that we should define the transition functions for any number of children: $\delta : \bigcup_{n=0}^{\infty} Q^n \times \Sigma \to 2^Q$. To achieve the latter, Brüggemann-Klein, Murata, and Wood [3], based on the work of Pair and Quere [30] and Takahashi [36], use regular string languages over Q to represent transitions. That is, the transition function δ is a mapping $\delta : Q \times \Sigma \to 2^{Q^*}$ such that $\delta(q, a)$ is a regular string language over Q.

Definition 3.3. 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 δ is a function $Q \times \Sigma \to 2^{Q^*}$ such that $\delta(q, a)$ is a regular string language over Q^* for every $a \in \Sigma$ and $q \in Q$.

A *run* of B on a tree t is a labeling $\lambda : \mathrm{Dom}(t) \to Q$ such that for every $v \in \mathrm{Dom}(t)$ with n children,

$$\lambda(v1) \cdots \lambda(vn) \in \delta(\lambda(v), \mathrm{lab}^t(v)).$$

Note that when v has no children, then the criterion reduces to $\varepsilon \in \delta(\lambda(v), \mathrm{lab}^t(v))$. So, we do not need an explicit definition of the I_σ's anymore. A run is *accepting* iff the root is labeled with a final state, that is, $\lambda(\varepsilon) \in F$. A tree is accepted if there is an accepting run for it. The set of all accepted trees is denoted by $L(B)$.

We illustrate the above definition with two examples. In the first example we represent the transition functions by regular expressions. In the second example we use logical formulas.

Example 3.4. (1) We continue Example 3.2. Inner nodes can now have an arbitrary number of children. Again, we define an automaton accepting exactly the circuits evaluating to 1. Define $B = (Q, \Sigma, \delta, F)$ with $Q = \{0, 1\}$, $F = \{1\}$, and

$$\begin{aligned}
\delta(0,0) &:= \delta(1,1) := \{\varepsilon\};\\
\delta(0,1) &:= \delta(1,0) := \emptyset;\\
\delta(0,\wedge) &:= (0+1)^*0(0+1)^*;\\
\delta(1,\wedge) &:= 1^*;\\
\delta(0,\vee) &:= 0^*;\\
\delta(1,\vee) &:= (0+1)^*1(0+1)^*.
\end{aligned}$$

Intuitively, B works as follows: B assigns 0 (1) to 0-labeled (1-labeled) leaves; B assigns a 1 to a $\wedge$-labeled node iff all its children are 1; B assigns a 0 to a $\vee$-labeled node iff all its children are 0. Finally, B accepts when the root is labeled with 1. In Figure 3, we give an example of a tree and an accepting run.

(2) Let p be the XPath expression `/a//b[/b]//a`. We construct an automaton accepting precisely the trees matching p. The set of states Q consists of all the subpatterns of p and $F = \{$`/a//b[/b]//a`$\}$. The transition function is defined in Figure 4. We use logical formulas to denote regular languages. For all states $q_1, \ldots, q_n \in Q$ the formula $\bigwedge_{i=1}^{n} q_i$ denotes the set of strings containing all the states q_i (and possibly some others). If φ_1 and φ_2 are formulas, then $\varphi_1 \vee \varphi_2$ denotes the union of the set of strings defined by φ_1 and φ_2; true denotes the set of all strings over Q. Clearly, these formulas can only define regular languages. Transitions that are not mentioned are empty.

The most intuitive way to interpret the automaton is to read the rules in a top-down way. The automaton starts at the root in state `/a//b[/b]//a`. A run started as such can only be valid if the root is labeled with a and one of the children matches `//b[/b]//a`. Further, a b-labeled node can only be in state `//b[/b]//a` if (1) it has a child that matches `//b[/b]//a`; (2) there are two children matching `/b` and `//a`, respectively; or, (3) there is one child matching `/b//a`. An a-labeled node can only be in state `//b[/b]//a` if one of its children match the pattern `//b[/b]//a`. The remaining rules are self-explanatory. We give an example in Figure 5. □

3.4 Relationship with ranked automata

Unranked trees can be encoded into binary ones in several ways. In Figure 6 we illustrate one such possibility. Intuitively, 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 other children are right descendants of the first child in the encoding. Whenever there is a right child but no left child, a # is inserted. Also, when there is only a left child, a # is inserted for the right child.

By using the encodings enc and dec of Figure 6 one obtains the following proposition.

Proposition 3.5. [34]

- *For every unranked NTA B there is a tree automaton A over binary trees such that $L(A) = \{enc(t) \mid t \in L(B)\}$.*
- *For every tree automaton A over binary trees there is an unranked NTA B such that $L(B) = \{dec(t) \mid t \in L(A)\}$.*

Although Proposition 3.5 provides a tool for transferring results from ranked to unranked trees, it does not deal with issues which are specific for unranked tree automata. The complexity of decision problems for NTAs, for instance, depends on the formalism used to represent the regular string languages $\delta(q, a)$ in the transition function. As there are many ways to represent regular string languages (logical formulas, automata with various forms of control, grammars, regular expressions,...), Proposition 3.5 does not offer immediate help. Let NTA($\mathcal{M}$) denotes the set of NTAs where the regular languages $\delta(q, a)$ are represented by elements in the class $\mathcal{M}$ (for instance, the class of NFAs).

A closer inspection of enc and dec reveals that the translation between binary and unranked tree automata is polynomial for NTA(NFA)'s. For this reason, the latter class can be seen as the default for unranked tree automata. Also the complexity of the membership problem for this class is tractable.

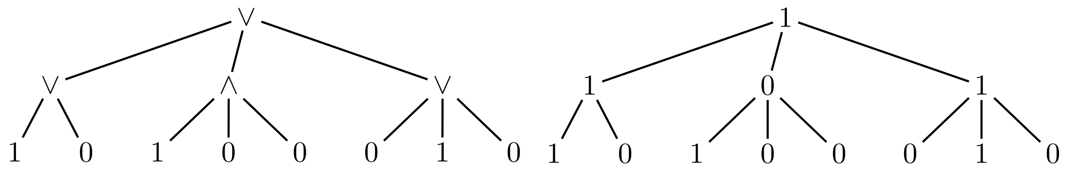


Figure 3: A tree and an accepting run of the automaton of Example 3.4(1).

$$
\begin{array}{lcl}
\delta(\texttt{/a//b[/b]//a}, a) & := & \texttt{//b[/b]//a} \\
\delta(\texttt{//b[/b]//a}, b) & := & \texttt{//b[/b]//a} \vee (\texttt{/b} \wedge \texttt{//a}) \vee \texttt{/b//a} \\
\delta(\texttt{//b[/b]//a}, a) & := & \texttt{//b[/b]//a} \\
\delta(\texttt{/b//a}, b) & := & \texttt{//a} \\
\delta(\texttt{//a}, a) & := & \text{true} \\
\delta(\texttt{//a}, b) & := & \texttt{//a} \\
\delta(\texttt{/b}, b) & := & \text{true}
\end{array}
$$

Figure 4: The automaton of Example 3.4(2) accepting `/a//b[//a]//a`.

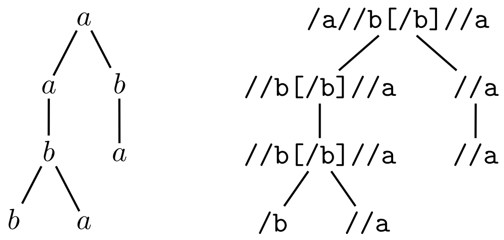


Figure 5: A tree and an accepting run of the automaton of Example 3.4(2).

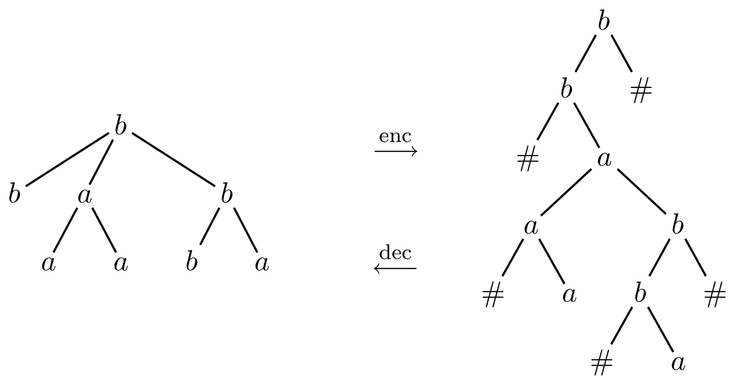


Figure 6: An unranked tree and its binary encoding.

Proposition 3.6. *Let $t \in \mathcal{T}_\Sigma$ and $B \in NTA(NFA)$. Testing whether $t \in L(B)$ can be done in time $\mathcal{O}(|t||B|^2)$.*

However, when using tree automata to obtain upper bounds on the complexity of problems related to XML, one sometimes needs to turn to more expressive formalisms. In [15], an upper bound on the complexity of the typechecking problem for structural recursion is obtained by a reduction to the emptiness problem of NTA(2AFA)'s. Here, 2AFA are two-way alternating string automata.

3.5 Schemas

Like extended context-free grammars form abstractions of DTDs, unranked tree automata are an abstraction of the various XML schema proposals.

Definition 3.7. A DTD is a tuple (d, s_d) where d is a function that maps Σ-symbols to regular expressions over Σ and $s_d \in \Sigma$ is the start symbol.

Example 3.8. As an example consider the following DTD (taken from [35]) describing a catalog of products

$$\begin{array}{lll} d(\mathit{catalog}) & := & \mathit{product}^* \\ d(\mathit{product}) & := & \mathit{name}? \cdot (\mathit{mfr\text{-}price} + \mathit{sale\text{-}price}) \cdot \\ & & \hfill \mathit{color}^* \end{array}$$

Here, $w?$ denotes $w + \varepsilon$.

The equivalent unranked tree automaton is defined next: $B = (Q, \Sigma, \delta, F)$ with $Q := \{\mathit{catalog}, \mathit{product}, \mathit{name}, \mathit{mfr\text{-}price}, \mathit{sale\text{-}price}, \mathit{color}\}$, $F := \{\mathit{catalog}\}$, for all $a \in \{\mathit{name}, \mathit{mfr\text{-}price}, \mathit{sale\text{-}price}, \mathit{color}\}$, $\delta(a, a) = \{\varepsilon\}$, and

$$\begin{array}{lll} \delta(\mathit{catalog}, \mathit{catalog}) & := & \mathit{product}^* \\ \delta(\mathit{product}, \mathit{product}) & := & \mathit{name}? \cdot (\mathit{mfr\text{-}price} + \\ & & \hfill \mathit{sale\text{-}price}) \cdot \mathit{color}^* \end{array}$$

The $\delta(q, a)$ that are not mentioned are empty. □

It is not so hard to see that for every DTD there is an equivalent unranked tree automaton. Moreover, the unranked tree automata are equivalent to the specialized DTDs of Papakonstantinou and Vianu[31] and the XDuce types of Hosoya and Pierce [10]. Lee, Mani, and Murata provide provide a comparison of XML schema languages based on formal language theory [13].

4 Related work

We briefly discuss a number of applications of unranked tree automata or related formalisms. We refer the interested reader to [26] for a more detailed overview of pattern languages based on tree automata.

Several researchers defined pattern languages for unranked trees that can be implemented by unranked tree automata: Neumann and Seidl develop a μ-calculus for expressing structural and contextual conditions on forests [21].[1] They show that their formalism can be implemented by push-down forest automata. The latter are special cases of unranked tree automata. Murata defines an extension of path expressions based on regular expressions over unranked trees [20]. Brüggemann-Klein and Wood consider caterpillar expressions [4]. These are regular expressions that in addition to labels can specify movement through the tree. Neven and Schwentick define a guarded fragment ETL of monadic second-order logic (MSO) whose combined complexity is much more tractable than that of general MSO [32, 24]. Expressiveness and complexity results on ETL are partly obtained via techniques based on unranked tree automata.

Neven and Schwentick define query automata [27]. These are two-way deterministic unranked tree automata that can select nodes in the tree. Query automata correspond exactly to the unary queries definable in monadic second-order logic. By a result of Gottlob and Koch they also correspond to the unary queries definable in monadic datalog [9].

In [22], an extension of the Boolean attribute grammars considered in [29] to unranked trees is defined. These also express precisely the unary queries in MSO. A translation of the region algebra, considered by Consens and Milo [5], into these attribute grammars drastically improves the complexity of the optimization problems of the former.

[1] A forest is a concatenation of unranked trees.

5 Discussion

We recalled the definition of unranked tree automata and provided some examples of applications in the context of XML. In addition to tree automata, some other formalisms regained attention in the area of databases:

Tree-walking. The tree-walking paradigm dates back from the early research on compilers and attribute grammars [1, 6]. Tree-walking automata are still considered in formal language theory as the connection with tree automata is still unknown [7, 8, 25]. In the context of XML, the tree-walking paradigm attracted attention as an abstraction of XML query languages [2, 17, 23] and streaming [33].

Infinite alphabets. The framework considered in this paper is limited in two ways. It assumes that the element names of XML documents are from a finite and known set and it ignores the data values in the leaf nodes and attributes of XML documents. For this reason, the work of Kaminski and Francez [11] on automata on infinite alphabets has been reexamined from an XML perspective [28, 23].

We mention that Miklau and Suciu obtained a translation of XPath expressions to ranked tree automata through an involved simulation [16]. The translation shows how the different XPath language constructs parameterize the complexity of the XPath containment problem. Although a direct translation of that XPath fragment to unranked automata is immediate, it is not clear at the moment whether there is an easy translation to unranked tree automata which gives the same parameterized results for containment. The main problem is the filter predicate as each predicate introduces a conjunction.

Acknowledgment

The author thanks Dan Suciu, Leonid Libkin, Jan Van den Bussche, Wim Martens, and Stijn Vansummeren for comments on an earlier version of this paper.

References

[1] A. V. Aho and J. D. Ullman. Translations on a context-free grammar. *Inform. and Control*, 19:439–475, 1971.

[2] G. J. Bex, S. Maneth, and F. Neven. A formal model for an expressive fragment of XSLT. *Information Systems*, 27(1):21–39, 2002.

[3] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1, april 3, 2001. Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001.

[4] A. Brüggemann-Klein and D. Wood. Caterpillars: A context specification technique. *Markup Languages*, 2(1):81–106, 2000.

[5] M. Consens and T. Milo. Algebras for querying text regions: Expressive power and optimization. *Journal of Computer and System Sciences*, 3:272–288, 1998.

[6] P. Deransart, M. Jourdan, and B. Lorho. *Attribute Grammars: Definition, Systems and Bibliography*, volume 323 of *Lecture Notes in Computer Science*. Springer, 1988.

[7] J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In J. Karhumki, H. Maurer, G. Paun, and G.Rozenberg, editors, *Jewels are forever, contributions to Theoretical Computer Science in honor of Arto Salomaa*, pages 72–83. Springer-Verlag, 1999.

[8] J. Engelfriet, H.J. Hoogeboom, and J.-P. van Best. Trips on trees. *Acta Cybernetica*, 14:51–64, 1999.

[9] G. Gottlob and C. Koch. Monadic datalog and the expresive power of languages for web information extraction. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 17–28. ACM Press, 2002.

[10] H. Hosoya and B. C. Pierce. Regular expression pattern matching for XML. In *Proceedings of 28th Symposium on Principles of Programming Languages (POPL 2001)*, pages 67–80. ACM Press, 2001.

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

[12] N. Klarlund, A. Moller, and M. I. Schwartzbach. The DSD schema language. In *Proceedings of the 3th ACM SIGSOFT Workshop on Formal Methods in Software Practice (FMSP 2000)*, 2000.

[13] D. Lee, M. Mani, and M. Murata. Reasoning about XML schema languages using formal language theor. Technical report, IBM Almaden Research Center, 2000. Log# 95071.

[14] S. Maneth and F. Neven. Structured document transformations based on XSL. In R. Connor and A. Mendelzon, editors, *Research Issues in Structured and Semistructured Database Programming (DBPL'99)*, volume 1949 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2000.

[15] W. Martens and F. Neven. Typechecking of top-down uniform unranked tree transducers. Manuscript.

[16] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 65–76, 2002.

[17] T. Milo, D. Suciu, and V. Vianu. Type checking for XML transformers. In *Proceedings of the Nineteenth ACM Symposium on Principles of Database Systems*, pages 11–22. ACM Press, 2000.

[18] M. Murata. Relax. `http://www.xml.gr.jp/relax/`.

[19] M. Murata. Data model for document transformation and assembly. In E. V. Munson, K. Nicholas, and D. Wood, editors, *Proceedings of the workshop on Principles of Digital Document Processing*, volume 1481 of *Lecture Notes in Computer Science*, pages 140–152, 1998.

[20] M. Murata. Extended path expressions for xml. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 126–137. ACM Press, 2001.

[21] A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In V. Arvind and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, pages 134–145. Springer, 1998.

[22] F. Neven. Extensions of attribute grammars for structured document queries. In R. Connor and A. Mendelzon, editors, *Research Issues in Structured and Semistructured Database Programming (DBPL'99)*, volume 1949 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2000.

[23] F. Neven. On the power of walking for querying tree-structured data. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 77–84. ACM Press, 2002.

[24] F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. In *Proc. 19th Symposium on Principles of Database Systems (PODS 2000)*, pages 145–156, 2000.

[25] F. Neven and T. Schwentick. On the power of tree-walking automata. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *International Colloquium on Automata, Languages and Programming (ICALP 2000)*, volume 1853 of *Lecture Notes in Computer Science*, pages 547–560. Springer, 2000.

[26] F. Neven and T. Schwentick. Automata- and logic-based pattern languages for tree-structured data. Unpublished, 2001.

[27] F. Neven and T. Schwentick. Query automata on finite trees. *Theoretical Computer Science*, 275:633–674, 2002.

[28] F. Neven, T. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. In J. Sgall, A. Pultr, and P. Kolman, editors, *Mathematical Foundations of Computer Science (MFCS 2001)*, volume 2136 of *Lecture Notes in Computer Science*, pages 560–572. Springer, 2001.

[29] F. Neven and J. Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. *Journal of the ACM*, 49(1), 2002.

[30] C. Pair and A. Quere. Définition et etude des bilangages réguliers. *Information and Control*, 13(6):565–593, 1968.

[31] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 35–46. ACM Press, 2001.

[32] T. Schwentick. On diving in trees. In *Proceedings of 25th Mathematical Foundations of Computer Science (MFCS 2000)*, pages 660–669, 2000.

[33] L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 53–64. ACM Press, 2002.

[34] D. Suciu. Typechecking for semistructured data. In *Proceedings of the 8th Workshop on Data Bases and Programming Languages (DBPL 2001)*, 2001.

[35] D. Suciu. The XML typechecking problem. *SIGMOD Record*, 31(1):89–96, 2002.

[36] M. Takahashi. Generalizations of regular sets and their application to a study of context-free languages. *Information and Control*, 27(1):1–36, 1975.

[37] M. Y. Vardi. Automata theory for database theoreticians. In *Proceedings of the Eighth ACM Symposium on Principles of Database Systems*, pages 83–92. ACM Press, 1989.

[38] V. Vianu. A web odyssey: From Codd to XML. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 1–15, 2001.