Streaming Tree Automata

Olivier Gauwin, Joachim Niehren, Yves Roos

Streaming Tree Automata

Olivier Gauwin Joachim Niehren Yves Roos
Mostrare project, INRIA and LIFL of Université de Lille 1, UMR CNRS 8022, Lille, France

Abstract

Streaming validation and querying of XML documents are often based on automata for tree-like structures. We propose a new notion of *streaming tree automata* in order to unify the two main approaches, which have not been linked so far: *automata for nested words* or equivalently *visibly pushdown automata*, and respectively *pushdown forest automata*.

Key words: tree automata, XML databases, query languages, streaming

1. Introduction

Streaming evaluation of XML documents means to process XML trees in document order. Streaming is useful in various XML applications requiring validation, typing, and querying. In this article, we subscribe to fundamental approaches to XML processing based on tree automata [2] that are complementary to standardization oriented formalisms such as DTDs [15], XPath [12] or XQuery [13,10].

Various notions of finite automata for tree-like structures have been proposed for streaming purpose. Tree walking automata are a traditional example [1], but they do not capture all regular tree languages [5] even when permitting pebbles [6]. Finite automata operating on linearizations of trees are another approach [18,17], but again, not all regular tree languages can be expressed. What is missing is the availability of a stack, as used in implementations of tree automata. This raises the question of how to characterize the particularities of such stacks.

This question has been approached in two independent research lines. The first line started in 2004 in the context of program verification, when Alur and Madhusudan [3] introduced *visibly pushdown automata* for this purpose. More recently, their relevance for XML streaming tasks was highlighted [14], and Alur [2] reformulated these into *nested word automata* (NWAs). The second line started as early as 1998, when Neumann and Seidl proposed *pushdown forest automata* (PFAs) for XML query evaluation. The similarity between NWAs and PFAs has been noticed only very recently [14]. In this paper, we formalize and prove the precise equivalence.

As a first contribution, we propose *streaming tree automata* (STAs) by reformulating runs of NWAs so that they directly operate on unranked trees (rather than encodings thereof). We believe that this reformulation is natural and useful elsewhere. As a second contribution, we use STAs in order to establish an equivalence between PFAs and NWAs. Our proof is based on the comparison of runs. This has the advantage to link run-based queries defined with such automata, too. These queries capture MSO-definable n-ary queries [8]. Query answering algorithms for PFAs [4] can be lifted easily to STAs too.

Email addresses: olivier.gauwin@inria.fr (Olivier Gauwin), www.grappa.univ-lille3.fr/~niehren (Joachim Niehren), yves.roos@lifl.fr (Yves Roos).

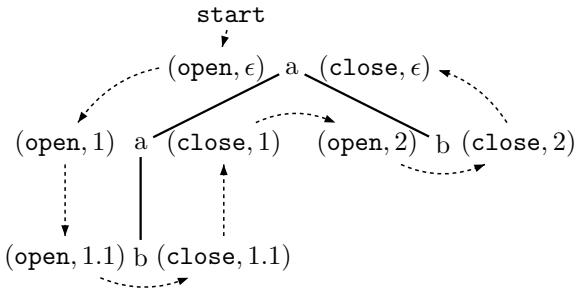


Fig. 1. The total ordering on $events(a(a(b), b))$

2. Trees, Events, and Runs of Automata

XML documents, and other forms of semi-structured data, are frequently abstracted into labeled unranked trees. Let Σ be a finite set of labels. An *unranked tree* $t \in T_\Sigma$ is either a constant $a \in \Sigma$ or a pair $a(t_1, \ldots, t_n)$, where $a \in \Sigma$ is a label, and $(t_1, \ldots, t_n)$ is a sequence of trees in T_Σ with $n \geq 1$. We write $nod(t) \subseteq \mathbb{N}^*$ for the set of nodes of a tree t. If t is the constant a then $nod(t) = \{\epsilon\}$. Otherwise, $nod(a(t_1, \ldots, t_n)) = \{\epsilon\} \cup \{i.\pi \mid \pi \in nod(t_i)\}$. Here, ϵ is the empty word standing for the *root* of t, and the word $i.\pi$ is the concatenation of letter i with word π. We write $lab^t(\pi)$ for the unique label of node π in t.

The preorder traversal over t produces a totally ordered set of *events*. Once initiated, it visits every node twice, once when opening (entering) the subtree it roots, and once when closing (leaving) it:

$$events(t) = \{\mathtt{start}\} \cup (\{\mathtt{open}, \mathtt{close}\} \times nod(t))$$

The unique predecessor $pred(e)$ for $e \in events(t)$ except $\{\mathtt{start}\}$ is the event preceding e in the preorder traversal of t, as illustrated in Fig. 1.

We will use a uniform syntax for different kinds of tree automata operating in document order. They all have two finite collections of states $stat = stat_\mathtt{n} \uplus stat_\mathtt{e}$ called node states and event states respectively, and subsets $init, final \subseteq stat_\mathtt{e}$ of initial and final states. Furthermore, they provide a finite set rul of rules whose forms may differ according to the concrete automata notion. Whenever useful, we will index the components of an automaton A by upper index A so that $A = (\Sigma^A, stat^A, init^A, final^A, rul^A)$.

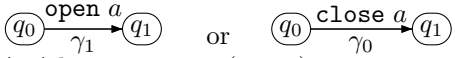
Runs of automata A on trees $t \in T_\Sigma$ are pairs $r = (r_\mathtt{e}, r_\mathtt{n})$ where $r_\mathtt{e} : events(t) \to stat_\mathtt{e}^A$ and $r_\mathtt{n} : nod(t) \to stat_\mathtt{n}^A$ assign states to events and nodes, such that $r_\mathtt{e}(\mathtt{start}) \in init^A$. Runs must be licensed by the rules of the automaton; what this means depends on the particular class. A run r is called successful if $r_\mathtt{e}(\mathtt{close}, \epsilon) \in final^A$. The language recognized by A, denoted by $L(A)$, is the set of all trees $t \in T_\Sigma$ on which A permits a successful run.

3. Streaming Tree Automata

Definition 1 *A* streaming tree automaton (STA) *is a tree automaton following the general notation above whose rules are terms of the following forms, where $q_0, q_1 \in stat_\mathtt{e}$, $a \in \Sigma$ and $\gamma_0, \gamma_1 \in stat_\mathtt{n}$:*

$$\mathtt{open}\ a\ q_0 \ \to\ q_1\ \gamma_1 \quad \textit{or} \quad \mathtt{close}\ a\ q_0\ \gamma_0 \ \to\ q_1$$

More formally, rules are tuples in $\{\mathtt{open}, \mathtt{close}\} \times \Sigma \times stat_\mathtt{e} \times stat_\mathtt{e} \times stat_\mathtt{n}$ written in a smoother syntax. In drawings, we will use the following alternative graphical notation for these rules:



An STA A licenses a run $(r_\mathtt{e}, r_\mathtt{n})$ on a tree t if the following are rules of A for all $\pi \in nod(t)$ with $a = lab^t(\pi)$:

$$\mathtt{open}\ \ a\ \ r_\mathtt{e}(pred(\mathtt{open}, \pi)) \ \to\ r_\mathtt{e}(\mathtt{open}, \pi)\ \ r_\mathtt{n}(\pi)$$

$$\mathtt{close}\ a\ r_\mathtt{e}(pred(\mathtt{close}, \pi))\ r_\mathtt{n}(\pi) \to r_\mathtt{e}(\mathtt{close}, \pi)$$

An example for a successful run of the STA in Fig. 3 on tree $a(a(b), b)$ is given in Fig. 4.

An STA is deterministic if it does not have two rules with the same left-hand side and has a unique initial state. Deterministic STAs can recognize all regular languages of unranked trees. The analogous result is well-known for NWAs. It can be shown by encoding (bottom-up and left-to-right) deterministic stepwise tree automata [8,9] into deterministic STAs.

Deterministic STAs are useful for streaming algorithms. They can be obtained by translating extended DTDs that are restrained competition and deterministic [14], so that running such STAs performs one-pass typing. They are equally useful for earliest query answering as shown in follow up work [11].

For illustration, we present the translation of DTDs to STAs. Given a deterministic DTD with alphabet Σ, we compute the collection of Glushkov automata $(G_a)_{a \in \Sigma}$ over Σ, which are deterministic finite automata for the regular expressions of the DTD [7]. Let $root \in \Sigma$ be the root symbol of the DTD.

From the collection of Glushkov automata, we construct a deterministic STA S recognizing the trees validated by the DTD. The states S unify the

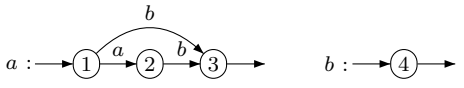


Fig. 2. Glushkov automata for DTD $a \to ab + b$ and $b \to \epsilon$

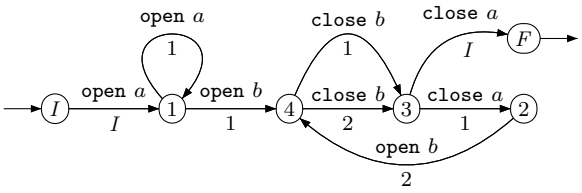


Fig. 3. The STA for the DTD in Fig. 2

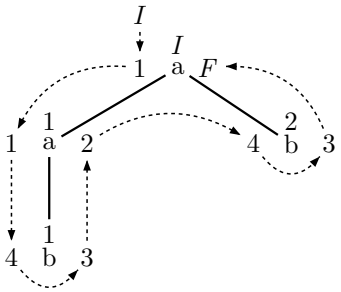


Fig. 4. Successful run of the STA in Fig. 3

states of all Glushkov automata and add a unique initial state I and a unique final state F:

$$stat^S = \uplus_{a \in \Sigma} stat^{G_a} \uplus \{I, F\}$$

The rules of the STA S are obtained systematically from those of the Glushkov automata according to the two following inference schemas:

$$\frac{q_0 \xrightarrow{b} q_1 \in rul^{G_a} \quad q_2 \in init^{G_b} \quad q_3 \in final^{G_b}}{\begin{array}{c}\texttt{open}\ b\ q_0 \ \to\ q_2\ q_0 \in rul^S \\ \texttt{close}\ b\ q_3\ q_0 \ \to\ q_1 \in rul^S\end{array}}$$

$$\frac{a = root \qquad q_0 \in init^{G_a} \qquad q_1 \in final^{G_a}}{\begin{array}{c}\texttt{open}\ a\ I \to q_0\ I \in rul^S \\ \texttt{close}\ a\ q_1\ I \to F \in rul^S\end{array}}$$

For instance, the STA drawn in Fig. 3 accepts valid documents for the DTD in Fig. 2.

A successful run on tree $a(a(b), b)$ is shown in Fig. 4. This construction preserves determinism, in that DTDs with deterministic Glushkov automata are translated to deterministic STAs.

4. Nested Word Automata

Nested word automata (NWAs) [2] are equal to STAs syntactically but run on nested words, so they have different semantics. We show that both semantics coincide modulo encoding unranked trees into nested words.

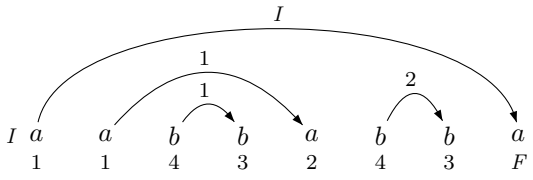


Fig. 5. Successful run of the NWA in Fig. 3

A nested word over Σ is a pair (w, E) where $w \in \Sigma^*$ is a word and $E \subseteq pos(w) \times pos(w)$ a set of forward edges without overlap. The set $pos(w)$ is the set of integers in $\{1, \ldots, |w|\}$ where $|w|$ is the length of w. We assume that every position in a nested word is adjacent to exactly one edge, and that for every edge, both adjacent positions have the same label. [1]

A run of an NWA A on a nested word (w, E) annotates all positions of $pos(w)$, the start position 0, and all edges in E by states, as illustrated by the example in Fig. 5. More precisely, a run of A as an NWA consists of two functions $r = (r_{\mathtt{e}}, r_{\mathtt{n}})$ with types $r_{\mathtt{e}} : pos(w) \cup \{0\} \to stat^A_{\mathtt{e}}$ and $r_{\mathtt{n}} : E \to stat^A_{\mathtt{n}}$. It is licensed by A if for all edges $(i, j) \in E$ adjacent to positions labeled by a, the following tuples belong to rul^A:

$$\texttt{open}\ a\ r_{\mathtt{e}}(i-1) \to r_{\mathtt{e}}(i)\ r_{\mathtt{n}}(i, j)$$
$$\texttt{close}\ a\ r_{\mathtt{e}}(j-1)\ r_{\mathtt{n}}(i, j) \to r_{\mathtt{e}}(j)$$

Unranked trees $t \in T_\Sigma$ can be encoded into nested words $nw(t) = (w, E)$ over Σ. For instance, the nested word for $a(a(b), b)$ is drawn in Fig. 5. More formally, let $e_1 \ \ldots \ e_n$ be the sequence of events in t except `start` in their total order. The word:

$$w = a_1 \ldots a_n$$

is the sequence of all $a_i \in \Sigma$ labeling the nodes of event e_i in t where $1 \leq i \leq n$. The edges link opening to closing events of the same node, i.e.:

$$E = \{(i, j) \mid \pi \in nod(t),\ e_i = (\texttt{open}, \pi),\ e_j = (\texttt{close}, \pi)\}$$

The function $I_{\mathtt{e}} : events(t) \to pos(nw(t)) \cup \{0\}$ with $I_{\mathtt{e}}(\texttt{start}) = 0$ and $I_{\mathtt{e}}(e_i) = i$ for all $1 \leq i \leq n$ is a bijection, as well as function $I_{\mathtt{n}} : nod(t) \to E$ with $I_{\mathtt{n}}(\pi) = (I_{\mathtt{e}}((\texttt{open}, \pi)), I_{\mathtt{e}}((\texttt{close}, \pi)))$. Thus, events of t correspond to positions of $nw(t)$ or 0 and

[1] More general definitions of nested words in the literature do permit dangling edges, internal positions, and unmatched labels, that we exclude here.

nodes of t to edges of $nw(t)$. The edges of t do not have immediate counterparts in $nw(t)$, but can be infered from the relations of positions in $nw(t)$ nevertheless.

Proposition 2 *Let A be an STA over Σ and $t \in T_\Sigma$ an unranked tree. A run (r_n, r_e) on $nw(t)$ is licensed by A as an NWA if and only if the run $(r_n \circ I_n, r_e \circ I_e)$ on t is licensed by A as an STA.*

As a consequence, the runs of A on t and $nw(t)$ correspond bijectively, and t is accepted by A as an STA if and only if $nw(t)$ is accepted by A as an NWA.

Nested words (w, E) encoding unranked trees satisfy the following restriction:

no hedges: there exists an edge $(1, |w|) \in E$.

Conversely, all nested words satisfying this condition encode some unranked tree. Every edge (i, j) in E corresponds to one node π of this tree, using the common label of i and j. As no overlap occurs, positions between i and j can be translated into a sequence of trees, defining the children of π. The no hedges condition ensures that this sequence of trees has a unique root.

5. Pushdown Forest Automata

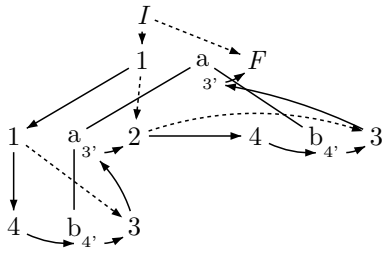
We recall PFAs from Neumann and Seidl [16] which operate on sequences of unranked trees (called forests there). We reformulate the original recursive definition of PFAs evaluators by formalizing a corresponding notion of runs. We restrict ourselves to tree languages, in that we define runs on trees only. This is no serious restriction, since our results extend easily to sequences of trees.

Definition 3 *A* pushdown forest automaton (PFA) *is a tree automaton following the general notation whose rules are of the following forms, where $q_0, q_1 \in stat_e$, $\gamma, \gamma_1 \in stat_n$ and $a \in \Sigma$:*

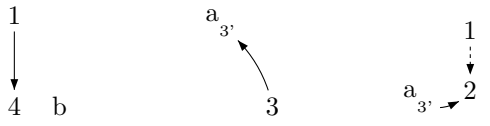
$$\texttt{down}\ a\ q_0 \to q_1 \quad \texttt{side}\ q_0\ \gamma \to q_1 \quad \texttt{up}\ a\ q_0 \to \gamma_1$$

Event states are originally called *forest states* and node states correspond to the original *tree states*. PFAs traverse trees in document order. When leaving a node π, two rules are used. First, an `up`-rule maps the node to some node state. Second, a `side`-rule assigns an event state to the closing event of the node. `up`-rules can be eliminated, but are kept here as in the original definition.

More formally, PFAs P permit runs $r = (r_e, r_n)$ on trees t if P contains the following rules for all nodes $\pi \in nod(t)$ with a label $a \in \Sigma$:



(a) Example of run



(b) `down` b $1 \to 4$ (c) `up` a $3 \to 3'$ (d) `side` 1 $3' \to 2$

Fig. 6. Run of a PFA

$$\texttt{down}\ a\ r_e(pred(\texttt{open}, \pi)) \to r_e(\texttt{open}, \pi)$$

$$\texttt{side}\ r_e(pred(\texttt{open}, \pi))\ r_n(\pi) \to r_e(\texttt{close}, \pi)$$

$$\texttt{up}\ a\ r_e(pred(\texttt{close}, \pi)) \to r_n(\pi)$$

Fig. 6(a) presents a run of a PFA on our example tree. The representation of rules is explained in Figs. 6(b), 6(c) and 6(d).

6. Translations between STA and PFA

We present polynomial time translations between weak STAs and PFAs and vice versa, which preserve runs up to simple correspondences and thus languages.

Weakness. Following [2], we call an STA weak if $stat_\texttt{n} = stat_\texttt{e}$ and all rules have the form $\texttt{open}\ a\ q_0 \to q_1\ q_0$ or $\texttt{close}\ a\ q_0\ q_1 \to q_2$. As proved in Theorem 1 of [2] for NWAs, every STA A is equivalent to some weak STA B of size at most $|B| = O(|stat^A_\texttt{e}| \cdot |stat^A_\texttt{n}|)$. To see this, let $stat^B_\texttt{n} = stat^B_\texttt{e} = stat^A_\texttt{e} \times stat^A_\texttt{n}$, with $init^B = init^A \times stat^A_\texttt{n}$ and $final^B = final^A \times stat^A_\texttt{n}$. The rules of B are derived from those of A according to the following two inference schemas.

$$\frac{\texttt{open}\ a\ q \to q_1\ \gamma_1 \in rul^A \quad \gamma_2 \in stat^A_\texttt{n}}{\texttt{open}\ a\ (q, \gamma_1) \to (q_1, \gamma_2)\ (q, \gamma_1) \in rul^B}$$

$$\frac{\begin{array}{c}\texttt{close}\ a\ q\ \gamma \to q_1 \in rul^A \\ \gamma_1, \gamma_2 \in stat^A_\texttt{n} \quad q_2 \in stat^A_\texttt{e}\end{array}}{\texttt{close}\ a\ (q, \gamma_1)\ (q_2, \gamma) \to (q_1, \gamma_2) \in rul^B}$$

From PFAs to weak STAs. We transform PFAs P into weak STAs $s(P)$ by removing intermediate tree states, identifying rules for `down` and `open`, and combining rules for `up` and `side` into `close`. Let $stat^{s(P)} = stat^P_{\mathtt{e}}$, $init^{s(P)} = init^P$, and $final^{s(P)} = final^P$, and let the following schemas define the rules of $s(P)$:

$$\frac{\mathtt{down}\ a\ q_0 \to q_1 \in rul^P}{\mathtt{open}\ a\ q_0 \to q_1\ q_0 \in rul^{s(P)}}$$

$$\frac{\mathtt{up}\ a\ q_1 \to \gamma_1 \in rul^P \qquad \mathtt{side}\ q_0\ \gamma_1 \to q_2 \in rul^P}{\mathtt{close}\ a\ q_1\ q_0 \to q_2 \in rul^{s(P)}}$$

From weak STAs to PFAs. Let A be a weak STA. We define a corresponding PFA $p(A)$ such that $s(p(A)) = A$. This shows that $p(A)$ and A recognize the same tree language. Let $stat^{p(A)}_{\mathtt{e}} = stat^A$ and $stat^{p(A)}_{\mathtt{n}} = \Sigma \times stat^A$, initial and final states remaining the same. The following inference schemas detail how the rules of $p(A)$ are inferred from A.

$$\frac{\mathtt{open}\ a\ q_0 \to q_1\ q_0 \in rul^A}{\mathtt{down}\ a\ q_0 \to q_1 \in rul^{p(A)}}$$

$$\frac{\mathtt{close}\ a\ q_0\ q_1 \to q_2 \in rul^A}{\begin{array}{c}\mathtt{up}\ a\ q_0 \to (a, q_0) \in rul^{p(A)} \\ \mathtt{side}\ q_1\ (a, q_0) \to q_2 \in rul^{p(A)}\end{array}}$$

Theorem 4 *Every PFA can be converted into an STA accepting the same language, and vice versa.*

PROOF. First, we prove that $L(s(P)) = L(P)$. This translation preserves the first function r_e of runs. Since $s(P)$ is weak, this function is sufficient to define a whole run of $s(P)$. Conversely, given a run of $s(P)$ on t, we can easily build the second function r_n as every `close` rule used in r_e is generated using an intermediate tree state. These translations preserve acceptance, so $L(P) = L(s(P))$.

Second, we show that for all weak STAs A, $s(p(A)) = A$. Recall that weakness can be assumed w.l.o.g. Translations of `open` and `down` rules are exactly symmetric. The double inclusion of `close` rules of A and $s(p(A))$ can be easily checked. Initial and final states are also preserved.

Thus, PFAs can be converted into weak STAs with fewer states so that the tree languages are preserved. Vice versa, there exists a language preserving translation which may increase the number of states by a factor of $|\Sigma|$.

The runs of STAs and corresponding PFAs assign the same event states to opening and closing events. This means that they define the same run-based queries, when selecting in event states only. This is illustrated in Fig. 6(a), by a run of the PFA corresponding to the STA of the previous example Fig. 4.

As a consequence, we can rely on the query answering algorithm for pushdown forest automata [4] for answering run-based weak STA queries. Removing the weakness limitation does not impose any problem. This way, we obtain a query answering algorithm for n-ary queries defined by STAs and thus NWAs.

Conclusion

We proposed STAs in order to prove the equivalence of NWAs and PFAs. The main advantage of STAs is that they directly operate on unranked trees rather than encodings thereof. We believe that deterministic STAs are the most natural tree automata notion for all kinds of XML streaming tasks. Adding weakness looks simpler but sometimes implies to put useless information in states.

Acknowledgments

The authors thank Anne-Cécile Caron, Luc Ségoufin and Sophie Tison for insightful work and discussions. This work was partially supported by the Enumeration project ANR-07-blanc.

References

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

[2] Rajeev Alur. Marrying words and trees. In *26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 233–242. ACM-Press, 2007.

[3] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *36th ACM Symposium on Theory of Computing*, pages 202–211. ACM-Press, 2004.

[4] Alexandru Berlea and Helmut Seidl. Binary queries for document trees. *Nordic Journal of Computing*, 11(1):41–71, 2004.

[5] Mikołaj Bojańczyk and Thomas Colcombet. Tree-walking automata do not recognize all regular languages. In *37th Annual ACM Symposium on Theory of Computing*, pages 234–243, New York, NY, USA, 2005. ACM-Press.

[6] Mikołaj Bojańczyk, Mathias Samuelides, Thomas Schwentick, and Luc Segoufin. Expressive power of pebbles automata. In *International Colloquium on Automata Languages and Programming (ICALP'06)*, Lecture Notes in Computer Science, pages 157–168. Springer Verlag, 2006.

[7] Anne Brüggemann-Klein. Regular expressions into finite automata. *Theor. Comput. Sci.*, 120(2):197–213, 1993.

[8] Julien Carme, Joachim Niehren, and Marc Tommasi. Querying unranked trees with stepwise tree automata. In *19th International Conference on Rewriting Techniques and Applications*, volume 3091 of *Lecture Notes in Computer Science*, pages 105 – 118. Springer Verlag, 2004.

[9] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available online since 1997. Revised October 2007: `http://www.grappa.univ-lille3.fr/tata`.

[10] Mary Fernandez, Philippe Michiels, Jérôme Siméon, and Michael Stark. XQuery streaming à la carte. In *23nd International Conference on Data Engineering*, pages 256–265, 2007.

[11] Olivier Gauwin, Anne-Cécile Caron, Joachim Niehren, and Sophie Tison. Complexity of earliest query answering with streaming tree automata. In *ACM SIGPLAN Workshop on Programming Language Techniques for XML (PLAN-X)*, January 2008. PLAN-X Workshop of ACM POPL.

[12] Ashish Kumar Gupta and Dan Suciu. Stream processing of XPath queries with predicates. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 419–430. ACM-Press, 2003.

[13] Christoph Koch, Stefanie Scherzinger, Nicole Schweikardt, and Bernhard Stegmaier. FluXQuery: An optimizing XQuery processor for streaming XML data. In *Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1309–1312, 2004.

[14] Viraj Kumar, Parthasarathy Madhusudan, and Mahesh Viswanathan. Visibly pushdown automata for streaming XML. In *16th international conference on World Wide Web*, pages 1053–1062. ACM-Press, 2007.

[15] Wim Martens, Frank Neven, and Thomas Schwentick. Which XML schemas admit 1-pass preorder typing? In *ICDT*, 2005.

[16] Andreas Neumann and Helmut Seidl. Locating Matches of Tree Patterns in Forests. In *Foundations of Software Technology and Theoretical Computer Science*, pages 134–145, 1998.

[17] Luc Segoufin and Cristina Sirangelo. Constant-memory validation of streaming XML documents against DTDs. In *Database Theory - ICDT 2007, 11th International Conference*, pages 299–313, 2007.

[18] Luc Segoufin and Victor Vianu. Validating streaming XML documents. In *Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 53–64, 2002.