Automata for Positive Core XPath Queries on Compressed Documents

Barbara Fila, Siva Anantharaman

Automata for Positive Core XPath Queries on Compressed Documents

Barbara Fila, Siva Anantharaman

LIFO - Université d'Orléans (France),
e-mail: {fila, siva}@univ-orleans.fr

Abstract. We present a method for evaluating positive unary queries expressed in terms of Core XPath axes on any XML document t, which may be presented in a fully or partially compressed form. To each Core XPath query of a certain basic type, we associate a word automaton; these automata run on the graph of dependency between the non-terminals of the minimal straightline regular tree grammar associated to the given document, or along complete sibling chains in this grammar. Any given Core XPath query can be decomposed into queries of the basic type, and the answer to the query can then be expressed as a sub-document of t, suitably labeled under the runs of such automata.

Keywords: Automata, Tree grammars, Dags, XML, Core XPath.

1 Introduction

Several algorithms have been optimized in the past, by using structures over dags instead of over trees. Tree automata are widely used for querying XML documents (e.g., [6, 7, 12, 13]); on the other hand, the notion of a compressed XML document has been introduced in [1, 5, 10], and a possible advantage of using dag structures for the manipulation of such documents has been brought out in [10]. It is legitimate then to investigate the possibility of using automata over dags instead of over trees, for querying compressed XML documents.

Our aim in this paper is to propose an approach based on word automata, for evaluating queries on XML documents that may not be in a fully unfolded tree format. With such an objective, we first present the notion of compression of documents, essentially as in [1, 5], and then define a *tree/dag* (*trdag*, for short) as a directed acyclic graph that may be partially or fully compressed (the terms 'trdag' and 'document' will be considered synonymous in the sequel). We adopt then the view that a trdag t is equivalent to a normalized straightline regular tree grammar $\mathcal{L}_t$ that one can naturally associate with t, cf. e.g., [2, 3]. From the grammar $\mathcal{L}_t$, we construct the graph of dependency $\mathcal{D}_t$ between its non-terminals, and also the *chiblings* (linear graphs formed of complete chains of sibling non-terminals) of $\mathcal{L}_t$. The word automata that we build below will run on $\mathcal{D}_t$, or on the chiblings of $\mathcal{L}_t$, rather than on the document t itself.

We shall only consider *positive* unary queries expressed in terms of Core XPath axes. (The view we adopt on compression allows us to define the various

axes of Core XPath on compressed documents, in a manner which does not modify their semantics on trees.) For evaluating any such query on any document t, we proceed as follows. We first break up the given query into basic sub-queries of the form Q= `//*[axis::`σ`]` where `axis` is a Core XPath axis of a certain type. To each such basic query Q, we associate a word automaton $\mathbf{A}_Q$. The automaton $\mathbf{A}_Q$ runs on the graph $\mathcal{D}_t$ when `axis` is non-sibling, and on the chiblings of $\mathcal{L}_t$ when `axis` is a sibling axis. An essential point in our method is that the runs of $\mathbf{A}_Q$ are guided by some well-defined semantics for the nodes traversed, indicating whether the current node answers Q, or is on a path leading to some other node answering Q. The automaton is not deterministic as such, but it is made effectively unambiguous by defining a suitable priority relation between its transitions, based on the semantics. A basic query Q can then be evaluated in one single top-down pass of $\mathbf{A}_Q$, under such an unambiguous run. An arbitrary positive unary Core XPath query can be evaluated on t by combining the answers to its various basic sub-queries, and its answer set is expressed as a sub-trdag of t, whose nodes get labeled in conformity with the semantics.

The paper is structured as follows: Section 2 presents the notion of trdags. In Section 3, we construct from any trdag t its normalized straightline regular tree grammar $\mathcal{L}_t$, as well as the dependency graph and the chiblings of $\mathcal{L}_t$; these will be seen as rooted labeled acyclic graphs (*rlags*, for short); the basic notions of Core XPath are also recalled. Section 4 is devoted to the construction of the word automata for any basic Core XPath query, based on the semantics, and an illustrative example. In Section 5, we prove that the runs of these automata, uniquely and effectively determined under a maximal priority condition, generate the answers to the queries. Section 6 shows how a non basic (composite, or imbricated) Core XPath query can be evaluated in a stepwise fashion. The appendix presents an algorithm for constructing the maximal priority run for any basic query automaton over any given document. Its complexity is polynomial on the number of nodes of the document.

2 Tree/Dags

Definition 1 *A tree/dag, or* trdag *for short, over an unranked alphabet Σ is a rooted dag (directed acyclic graph) $t = (Nodes(t), Edges(t))$, where:*

- every node $u \in Nodes(t)$ has a name $\in \Sigma$, denoted $name(u)$,

- and the edges going out of any node are ordered.

Given any node u on a trdag t, the notion of the sub-trdag of t rooted at u is defined as usual (it can be identified with u). If v is any node, $\gamma(v) = u_1 \dots u_n$ will denote the *string* of all its not necessarily distinct *children* nodes; for every $1 \leq i \leq n$, the i-th outgoing edge from v to its *i-th child* node $u_i \in \gamma(v)$ will be denoted as $\mathbf{e}(v, i)$; we shall also write then $v \xrightarrow{i} u_i$. For any node u other than the root node of t, we set: $Parents(u) = \{v \in Nodes(t) \mid u \text{ is a child of } v\}$.

A trdag t is said to be a *tree* iff for every node u on t other than the root, $Parents(u)$ is a singleton. For any trdag t, we define the set $Pos(t)$ as the set of all the positions $pos_t(u)$ of all its nodes u, these being defined recursively, as

follows: if u is the root node on t, then $pos_t(u) = \epsilon$, otherwise, $pos_t(u) = \{\alpha.i \mid \alpha \in pos_t(v), v \text{ is a parent of } u, u \text{ is an i-th child of } v\}$. The elements of $Pos(t)$ are words over natural integers, and the set $Pos(t)$ is totally ordered under the usual lexicographic ordering on such words, that we shall denote by $\leq_l$, and refer to as the *document ordering* on t.

Definition 2 *Given two trdags t and t' over an alphabet Σ, a* bisimilarity relation *between t and t' is a relation $\sim$ between $Nodes(t) = V$ and $Nodes(t') = W$, satisfying the following conditions:*

(i) $root(t) \sim root(t')$;

(i) the relation $\sim$ is compatible with the document orderings on t, t'*:*
if $v_1, v_2 \in V$, $w_1, w_2 \in W$ are such that $v_i \sim w_i, i = 1, 2$,
then $v_1 \leq_l v_2$ on t if and only if $w_1 \leq_l w_2$ on t';

(iii) for every $v \in V$ there exists at least one $w \in W$ with $v \sim w$,
and for every $w \in W$ there exists at least one $v \in V$ with $v \sim w$;

(iv) if $v \sim w$ then $name(v) = name(w)$;

(v) if $v \sim w$, then for all i, there exists $v \xrightarrow{i} v'$ on $Edges(t)$ if and only if there exists $w \xrightarrow{i} w'$ on $Edges(t')$, such that *$v' \sim w'$.*

A trdag is said to be a *tdag*, or *fully compressed*, if in addition to the conditions of Definition 1 above, the following condition is satisfied too: *no two distinct subgraphs are bisimilar.* Otherwise it is said to be *partially compressed.* A tree over Σ which is bisimilar to a trdag t is said to be its *tree equivalent.* For example, the tree to the left of Figure 1 is bisimilar to the partially compressed trdag to the right, and also to the fully compressed tdag to the middle.

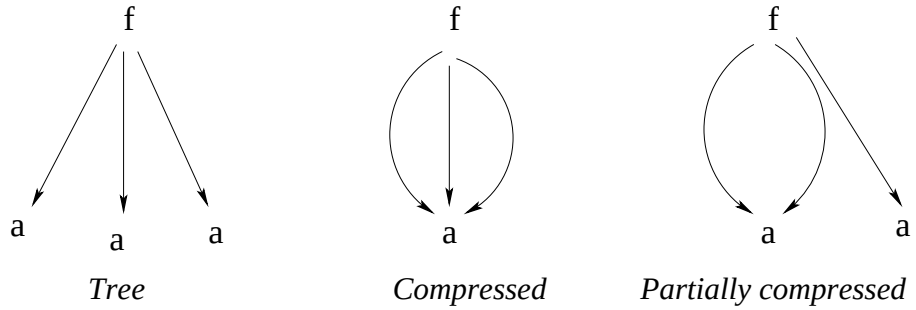


Fig. 1. tree, tdag, and trdag

Remark 1. It is possible to redefine the notion of (bottom-up) run of a tree automaton **A** on a trdag t, by adding assignments of states to the edges of t, in such a way that **A** accepts t in this extended sense, if and only if **A** accepts the tree-equivalent of t under a run in the classical sense. For details, see [4].

3 Querying Compressed Documents: Preliminaries

Given a trdag t, one can naturally construct a regular tree grammar associated with t, which is *straightline* (cf. [3]), in the sense that there are no cycles on

the dependency relations between its non-terminals, and each non-terminal produces exactly one sub-trdag of t. Such a grammar will be denoted as $\mathcal{L}_t$, if it is *normalized* in the following sense:

(i) for every non-terminal A_i of $\mathcal{L}_t$, there is exactly one production of the form $A_i \to f(A_{j_1}, \ldots, A_{j_k})$, where $i < j_r$ for every $1 \leq r \leq k$; we shall then set $Sons(A_i) = \{A_{j_1}, \ldots, A_{j_k}\}$, and $symb_{\mathcal{L}_t}(A_i) = f$;

(ii) the number of non-terminals is the number of nodes on t.

Such a normalized grammar $\mathcal{L}_t$ is uniquely defined up to a renaming of the non-terminals. For instance, for the trdag t to the left of Figure 2 we get the following normalized grammar:

$A_1 \to f(A_2, A_3, A_4, A_5, A_2), \quad A_2 \to c, \quad A_3 \to a(A_5), \quad A_4 \to b, \quad A_5 \to b.$

Such a grammar is easily constructed from t, for instance by using a standard algorithm which computes the 'depth' of any node (as the maximal distance from the root), to number the non-terminals so as to satisfy condition (i) above.

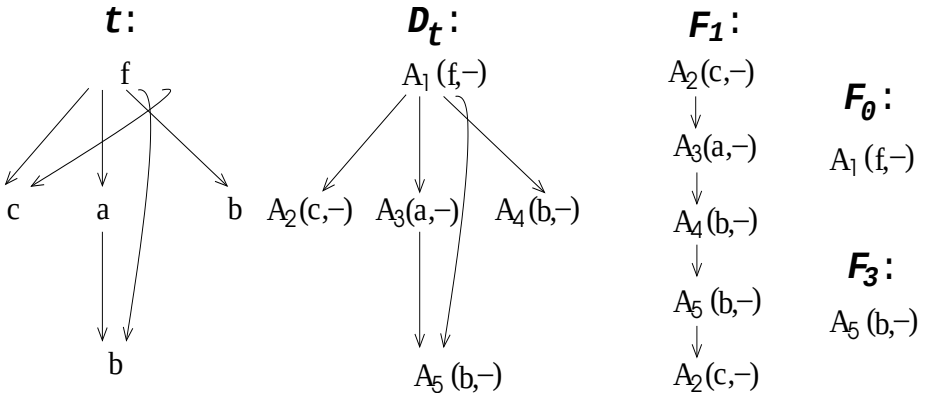


Fig. 2. trdag t, associated rlag $\mathcal{D}_t$, and chiblings of $\mathcal{L}_t$

The *dependency graph* of the normalized grammar $\mathcal{L}_t$ associated with t, and denoted as $\mathcal{D}_t$, consists of nodes named with the non-terminals $A_i, 1 \leq i \leq n$, and *one single* directed arc from any node A_i to a node A_j whenever A_j is a son of A_i. The root of $\mathcal{D}_t$ is by definition the node named A_1. The notion of *Sons* of the nodes on $\mathcal{D}_t$ is derived in the obvious way from that defined above on $\mathcal{L}_t$.

Furthermore, to any production $A_i \to f(A_{j_1}, \ldots, A_{j_k})$ of $\mathcal{L}_t$, we associate a rooted linear graph composed of k nodes respectively named $A_{j_1}, \ldots, A_{j_k}$, with root at A_{j_1} and such that for all $l \in \{2, \ldots, k\}$ the node named A_{j_l} is the son of the node named $A_{j_{l-1}}$. This graph will be called the *chibling* of $\mathcal{L}_t$ associated with the (unique) A_i-production; it is denoted as $\mathcal{F}_i$. We also define a further chibling denoted $\mathcal{F}_0$, as the linear graph with a single node named A_1, where A_1 is the axiom of $\mathcal{L}_t$

In the sequel, we designate by $\mathcal{G}$ either $\mathcal{D}_t$ or any of the chiblings $\mathcal{F}$ of $\mathcal{L}_t$. We complete any of these acyclic graphs $\mathcal{G}$ into a rooted labeled acyclic graph (*rlag*, for short), by attaching to each node u on $\mathcal{G}$, with $name(u) = A_i$, a label denoted $label(u)$, and defined as $label(u) = (symb_{\mathcal{L}_t}(A_i), -)$; cf. Figure 2.

3.1 Positive Core XPath Queries on trdags

In this paper we restrict our study to positive Core XPath queries on trdags. Recall that the Core XPath is the navigational segment of XPath, and is based on the following axes of XPath (cf. [15]): `self, child, parent, ancestor, descendant, following-sibling, preceding-sibling, sibling`. A location expression is defined as a predicate of the form `[axis::b]`, where `axis` is one of the above axes, and b is a symbol of Σ. Given any trdag t over Σ, a context node u on t and $b \in \Sigma$, the semantics for `axis` is defined by evaluating this predicate at u. The semantics for the axes `self, child, descendant` are easily defined, exactly as on trees (cf. [15]). For defining the semantics of the remaining axes, we first recall that $Parents(u) = \{v \in Nodes(t) \mid u \text{ is a child of } v\}$.

Definition 3 *Given a context node u on a trdag t, and $b \in \Sigma$:*

i) `[parent::b]` *evaluates to* true *at u, if and only if there exists a b-named node in $Parents(u)$;*

ii) `[ancestor::b]` *evaluates to* true *at u, iff either* `[parent::b]` *evaluates to* true *at u, or there exists a node $v \in Parents(u)$ such that* `[ancestor::b]` *evaluates to* true *at v;*

iii) `[following-sibling::b]` *evaluates to* true *at u, iff there exists a b-named node u', and a node v on t such that $\gamma(v)$ is of the form $...u...u'...$;*

iv) `[preceding-sibling::b]` *evaluates to* true *at u, iff there exists a b-named node u', and a node v on t such that $\gamma(v)$ is of the form $...u'...u...$;*

v) `[sibling::b]` *evaluates to* true *at u iff either* `[following-sibling::b]` *or* `[preceding-sibling::b]` *(or both) evaluate(s) to* true.

The semantics for the axes `descendant-or-self` and `ancestor-or-self` are then deduced in the obvious manner. And finally, for any set S of nodes on t, the sets of nodes `following(S)` and `preceding(S)` are defined formally, following [1], as follows:

`following(S)` =
`descendant-or-self(following-sibling(ancestor-or-self(S)))`,

`preceding(S)` =
`descendant-or-self(preceding-sibling(ancestor-or-self(S)))`.

Remark 2. i) Unlike on a tree, the `ancestor, descendant, following, self` and `preceding` axes do *not* partition the set of nodes on a compressed document.

ii) One can always switch from the format '`axis(`b`)`' to the format '`axis::`b'. For instance: for any b, the set of nodes `ancestor-or-self(b)` is the answer to the query //*[`descendant-or-self::b`].

Here are some easy consequences of our definitions.

Proposition 1 *(1) For any set of nodes X on any document (trdag) t, we have:*

`ancestor-or-self`$(X) = \bigcup_{x \in X}\{$ `//*[descendant::`x *or* `self::`x`]` $\}$
`descendant-or-self`$(X) = \bigcup_{x \in X}\{$ `//*[ancestor ::`x *or* `self::`x`]` $\}$
`following-sibling`$(X) = \bigcup_{x \in X}\{$ `//*[preceding-sibling::`x`]` $\}$
`preceding-sibling`$(X) = \bigcup_{x \in X}\{$ `//*[following-sibling::`x`]` $\}$

(2) For any trdag t, and any node with name b on t, we have:

(i) `//*[preceding::`b`] =`

$$\bigcup_{u, name(u)=b} \{\ \texttt{descendant-or-self(following-sibling(}\ \texttt{//*[descendant::}b\ \textit{or}\ \texttt{self::}b\texttt{]))}\}$$

(ii) `//*[following::`b`] =`

$$\bigcup_{u, name(u)=b} \{\ \texttt{descendant-or-self(preceding-sibling(}\ \texttt{//*[descendant::}b\ \textit{or}\ \texttt{self::}b\texttt{]))}\}$$

We shall refer to the axes `self, child, descendant, parent, ancestor, preceding-sibling, following-sibling` as *basic*. A basic Core XPath query is defined as a query of the form `//*[axis::`σ`]`, where `axis` is a basic axis. More generally, the queries we propose to evaluate on trdags are defined formally as the expressions Q generated by the following grammar, where σ stands for any node name on the documents:

A ::= `self` | `child` | `descendant` | `parent` | `ancestor` | `preceding-sibling` | `following-sibling`

Q ::= `//*[A::`σ`]` | `/*[A::`σ`]` | Q *or* Q | Q *and* Q | `//*[A::*[Q]]`

Core XPath queries Q of this format will be said to be in *standard form*. It is not hard to translate any positive Core XPath query into one in standard form; e.g., the query `//`c`[following-sibling::`g`]/`d is equivalent to `//*[self::`d`] and //*[parent::*[//*[self::`c`] and //*[following-sibling::`g`]]]`, in standard form. The components in a disjunction (resp. conjunction) under a '`*`' can be evaluated separately: indeed, `//*[A::`x *conn* `A'::`x'`]` is equivalent to `//*[A::`x`]` *conn* `//*[A'::`x'`]`, where *conn* is *or* (resp. *and*).

We shall be henceforth considering positive queries in the format `//*[A::`σ`]`, essentially; note that for evaluating a query of the form `/*[A::`σ`]`, one can retain the answers for `//*[A::`σ`]` that are at depth 1.

4 Automata for the Basic Queries

4.1 The Semantics of the Approach

We first consider basic Core XPath queries. Composite or imbricated queries will subsequently be evaluated in a stepwise fashion; see Section 6.

To any basic query $Q =$ `//*[axis::`σ`]`, we shall associate a word automaton (actually a transducer), referred to as $\mathbf{A}_Q$. It will run top-down, on the rlag $\mathcal{D}_t$ if `axis` is non-sibling, and on each of the chiblings $\mathcal{F}$ of $\mathcal{L}_t$ otherwise. In either case, a run will attach, to any node traversed, a pair of the form (ℓ, x), where the component ℓ of the pair has the intended semantics of selection or not, by Q, of the corresponding node on t, and the component x will be a 1 or 0, with the intended semantics that $x = 1$ iff the corresponding node on t has a descendant answering Q. At the end of the run, $label(u)$, at any node u of $\mathcal{D}_t$, will be replaced by a new label derived from the ll-pairs attached to u by the run.

To formalize these ideas, we introduce a set of new symbols $L = \{s, \eta, \top, \top'\}$ referred to as *llabels* (the term 'llabel' is used so as to avoid confusion with the

term label). We define *ll-pairs* as elements of the set $L \times \{0,1\}$, and the states of $\mathbf{A}_Q$ as elements of the set $\{init\} \cup (L \times \{0,1\})$. For any Q, the automaton $\mathbf{A}_Q$ is over the alphabet $\Sigma \cup \{s, \eta\}$, has $\{init\}$ as its initial state, and has no final state. The set Δ_Q of transitions of $\mathbf{A}_Q$ will consist of rules of the form $(q, \tau) \to q'$ where $q \in \{init\} \cup (L \times \{0,1\})$, $q' \in (L \times \{0,1\})$, and $\tau \in \Sigma \cup \{s, \eta\}$.

For any rlag $\mathcal{G}$, we define a function $llab\colon Nodes(\mathcal{G}) \to \Sigma \cup \{s, \eta\}$, by setting $llab(u) = \pi_1(label(u))$, the first component of $label(u)$. The automaton $\mathbf{A}_Q$ associated to a basic query $Q =$`//*[axis::`σ`]` will run *top-down* on the rlag $\mathcal{G}$, where $\mathcal{G}$ is $\mathcal{D}_t$ if `axis` is a basic non-sibling axis, and $\mathcal{G}$ is any chibling $\mathcal{F}$ of $\mathcal{L}_t$ if `axis` is a basic sibling axis. A *run* of $\mathbf{A}_Q$ on $\mathcal{G}$ is a map $r\colon Nodes(\mathcal{G}) \to (L \times \{0,1\})$, such that, for every $u \in Nodes(\mathcal{G})$, the following holds:

- if u is $root_{\mathcal{G}}$, then the rule $(init, llab(u)) \to r(u)$ is in Δ_Q;
- otherwise, for every $v \in \gamma(u)$ the rules $(r(u), llab(v)) \to r(v)$ are all in Δ_Q.

(Note: when `axis` is non-sibling, this amounts to requiring that, for any node v, the state $r(v)$ must be in conformity with the states $r(u)$ for *every* parent node u of v, with respect to the rules in Δ_Q.)

From the run of the automata $\mathbf{A}_Q$ and from the labels it attaches to the nodes of $\mathcal{D}_t$, we will also deduce, at every node u of t, a well-determined ll-pair as label at u (note: $Nodes(t), Nodes(\mathcal{D}_t)$ are in natural bijection). The ll-pairs thus attached to the nodes of t will have the following semantics (where x stands for the name of the node on t, corresponding to the 'current' node on $\mathcal{D}_t$):

- $(\top', 1) : x = \sigma$, current node on t is selected by (i.e., is an answer for) Q;
- $(\top, 1) : x = \sigma$, current node is *not* selected, but has a selected descendant;
- $(\top, 0) : x = \sigma$, current node is *not* selected, and has *no* selected descendant;
- $(s, 1) \ : x \neq \sigma$, current node is selected;
- $(\eta, 1) : x \neq \sigma$, current node is *not* selected, but has a selected descendant;
- $(\eta, 0) : x \neq \sigma$, current node is *not* selected, and has *no* selected descendant.

Only the nodes on $\mathcal{D}_t$, to which the run of A_Q associates the labels $(s, 1)$ or $(\top', 1)$, correspond to the nodes of t that will get selected by the query Q. The ll-pairs with boolean component 1 will label the nodes of $\mathcal{D}_t$ corresponding to the nodes of t which are on a path to an answer for the query Q; thus the automata $\mathbf{A}_Q$ will have *no* transitions from any state with boolean component 0 to a state with boolean component 1. Moreover, with a view to define runs of such automata which are unique (or unambiguous in a sense that will be presently made clear), we define some *priority* relations between the ll-pairs:

$$(\eta, 0) > (\eta, 1) > (s, 1), \qquad \text{and} \qquad (\top, 0) > (\top, 1) > (\top', 1).$$

A run of the automaton $\mathbf{A}_Q$ will label any node u on $\mathcal{G}$ only with an ll-pair of one of the two groups $\{(\top, 0), (\top, 1), (\top', 1)\}$ or $\{(\eta, 0), (\eta, 1), (s, 1)\}$, and this group will be determined by $llab(u)$.

For ease of presentation, we agree to set $\eta' := s$, and often denote either of the above two groups of ll-pairs under the uniform notation $\{(l, 0), (l, 1), (l', 1)\}$, where $l \in \{\eta, \top\}$, with the ordering $(l, 0) > (l, 1) > (l', 1)$.

We shall construct a run r of $\mathbf{A}_Q$ on $\mathcal{G}$ that will be uniquely determined by the following *maximal priority* condition:

(**MP**): at any node v on $\mathcal{G}$, $r(v)$ is the maximal ll-pair $(\imath, x)$ for the ordering $>$ in the group $\{(l,0),(l,1),(l',1)\}$ determined by $llab(v)$, such that $\mathbf{A}_Q$ contains a transition rule of the form $(r(u), llab(v)) \rightarrow (\imath, x)$, for *every* parent u of v.

Such a run will assign a label with boolean component 1 only to the nodes corresponding to those of the minimal sub-trdag t containing the root of t and all the answers to Q on t.

4.2 Re-labeling of $\mathcal{D}_t$ by the Runs of $\mathbf{A}_Q$

We first consider a non-sibling basic query Q on a given document t, and a given run r of the automaton $\mathbf{A}_Q$ on the $\mathcal{D}_t$; at the end of the run, the nodes on $\mathcal{D}_t$ will get re-labeled with new ll-pairs, computed as below for every $u \in Nodes(\mathcal{D}_t)$:

$$lab_r(u) = (s,1) \text{ iff } r(u) \in \{(s,1),(\top',1)\},$$
$$lab_r(u) = (\eta,1) \text{ iff } r(u) \in \{(\eta,1),(\top,1)\},$$
$$lab_r(u) = (\eta,0) \text{ iff } r(u) \in \{(\eta,0),(\top,0)\}.$$

The rlag obtained in this manner from $\mathcal{D}_t$, following the run r and the associated re-labeling function lab_r, will be denoted as $r(\mathcal{D}_t)$.

For a basic query Q over a sibling axis, the situation is a little more complex, because several different nodes on one chibling of $\mathcal{L}_t$ can have the same name (non-terminal), or several different chiblings can have nodes named by the same non-terminal, or both. Thus the runs of $\mathbf{A}_Q$ on the various chiblings of $\mathcal{L}_t$ will in general assign more than one ll-pair to a node of $\mathcal{D}_t$. We therefore proceed as follows: for every complete set $\widehat{r}$ of runs of $\mathbf{A}_Q$, one run on each chibling $\mathcal{F}$, we will define $\widehat{r}(\mathcal{D}_t)$ as the re-labeled rlag derived from $\mathcal{D}_t$, under $\widehat{r}$. With that purpose we first associate to $\widehat{r}$ and any $u \in Nodes(D_t)$, a set of ll-pairs:

$$ll_{\widehat{r}}(u) = \bigcup_{r_{\mathcal{F}} \in \widehat{r}} \{r_{\mathcal{F}}(v) \mid v \in Nodes(\mathcal{F}), \text{ and } name(v) = name(u)\}$$

where $r_{\mathcal{F}}$ is a run of $\mathbf{A}_Q$ on a chibling $\mathcal{F}$; we then derive, at each node of $\mathcal{D}_t$ a unique ll-pair in conformity with the semantics of our approach, by using the following function:

$$\lambda_{\widehat{r}}(u) = s \iff ll_{\widehat{r}}(u) \cap \{(s,1),(\top',1)\} \neq \emptyset,$$
$$\lambda_{\widehat{r}}(u) = \eta \iff ll_{\widehat{r}}(u) \cap \{(s,1),(\top',1)\} = \emptyset.$$

From $\mathcal{D}_t$ and this function $\lambda_{\widehat{r}}$, we next derive an rlag $\lambda_{\widehat{r}}(\mathcal{D}_t)$ by re-labeling each node u on $\mathcal{D}_t$ with the pair $(\lambda_{\widehat{r}}(u), -)$. And finally we define $\widehat{r}(\mathcal{D}_t)$ as the rlag obtained from $\lambda_{\widehat{r}}(\mathcal{D}_t)$, by running on it the automaton for the basic non-sibling query `//*[self::`s`]`, as indicated at the beginning of this subsection. In practical terms, such a run amounts in essence to setting, as the second component of $label(u)$ at any node u, the boolean 1 iff u is on a path to some node with s as its llab, and 0 otherwise. All these details are illustrated with an example in the following subsection.

4.3 The Automata

We first present the automata for the basic queries `//*[self::`σ`]` and for `//*[following-sibling::`σ`]`, and give an illustrative example using the former for $\sigma = s$, and the latter for $\sigma = b$. The automata for the other basic queries are given after the example.

• Automata: for `//*[self::`σ`]` and for `//*[following-sibling::`σ`]`

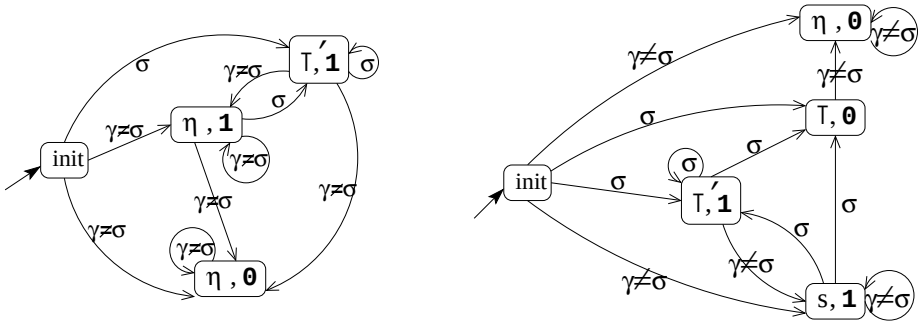


Figure 3 below illustrates the evaluation of $Q =$ `//*[following-sibling::`b`]`, on the trdag t of Figure 2. The sub-trdag of t, formed of nodes whose labels have boolean component 1, contains all the answers to Q on t.

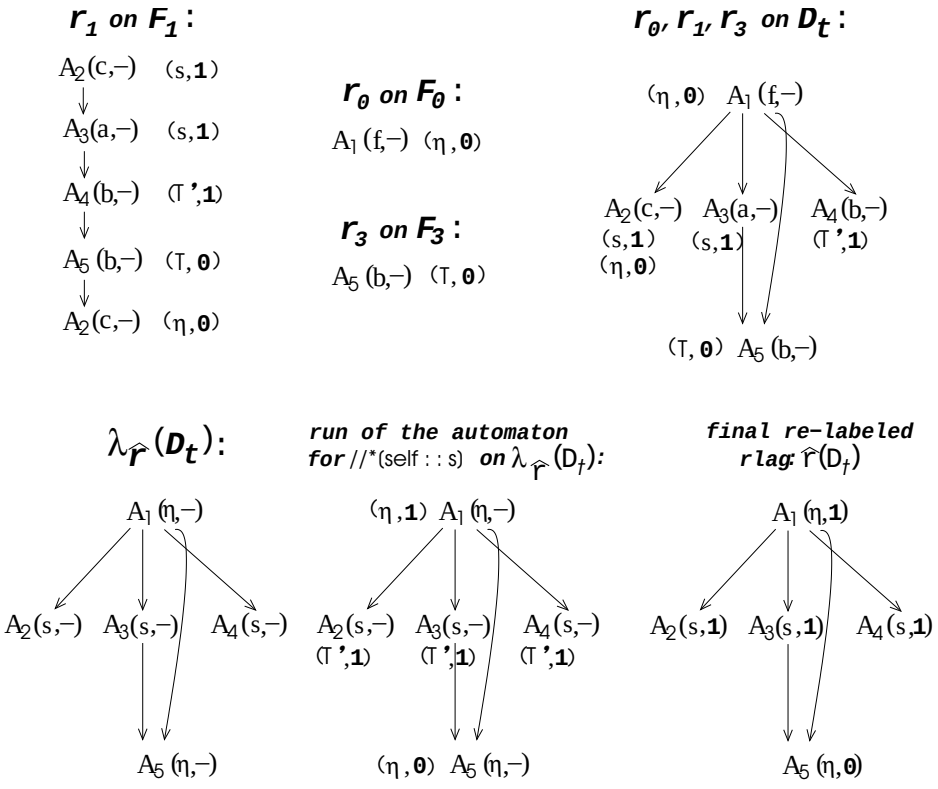
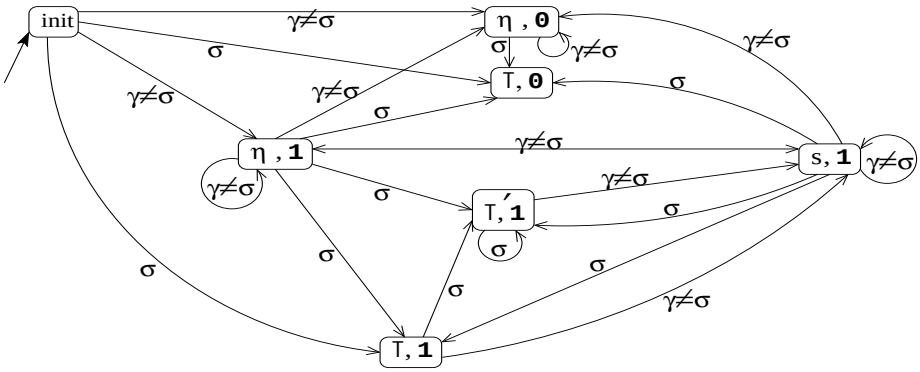
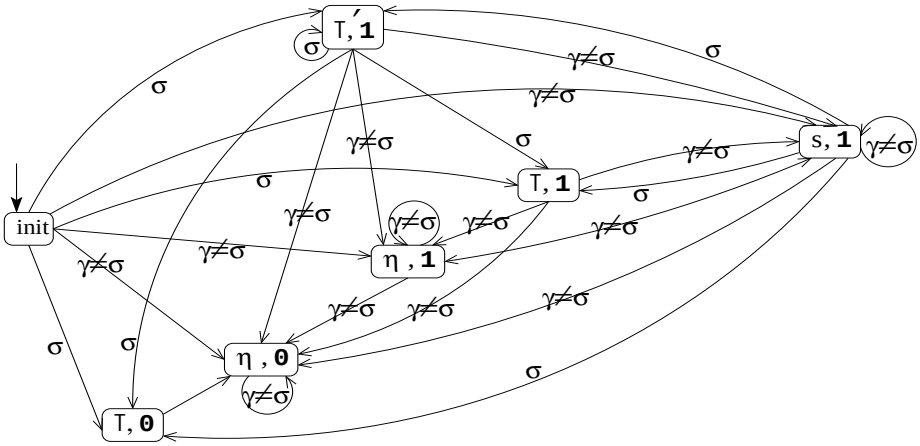


Fig. 3.

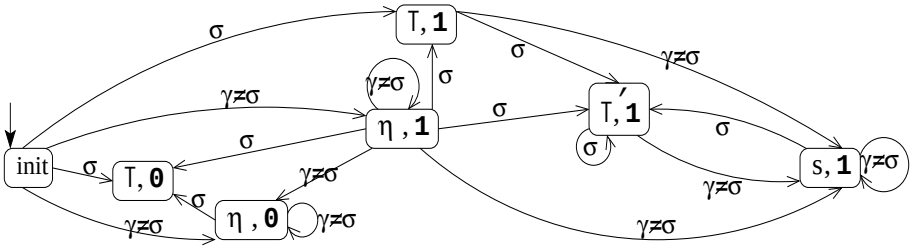
• Automaton for the query `//*[parent::`σ`]`



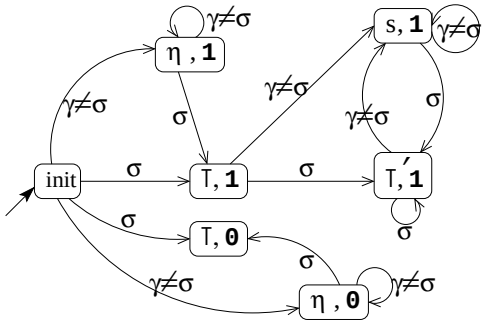
• Automaton for the query `//*[child::`σ`]`



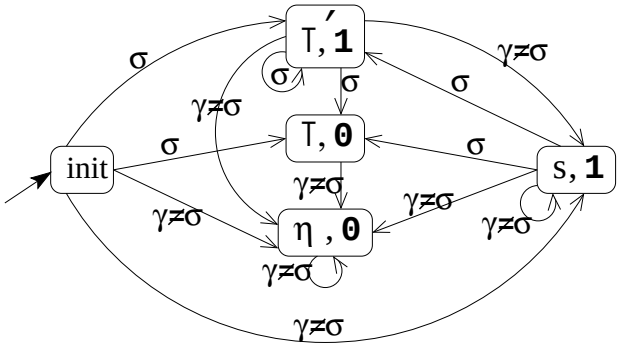
• Automaton for the query `//*[ancestor::`σ`]`



• Automaton for the query `//*[preceding-sibling::`σ`]`



• Automaton for the query `//*[descendant::`σ`]`



A few words on some of the automata by way of explanation. First, the reason why the automaton for `self` does *not* have the states $(\top, 0), (\top, 1), (s, 1)$: for $(\top, 0), (\top, 1)$, by the semantics of subsection 4.1 we must have $x = \sigma$, where x is the name of the current node on t, but then the query `//*[self::`σ`]` should select the current node, so one cannot be at such a state; as for $(s, 1)$, the reasoning is just the opposite. Next, the reason why the automaton for `descendant` does *not* have the states $(\eta, 1), (\top, 1)$: if the semantics attribute one of these pairs to any node u, that would mean the node u has a selected descendant u'; which means that u' has some σ-descendant node, which would then be a σ-descendant for u too, so Q should select u.

5 Maximal Priority Runs of Basic Query Automata

Note that the following properties, required by our semantics of subsection 4.1, hold on the automata $\mathbf{A}_Q$ constructed above, for any basic Core XPath query $Q =$ `//*[axis::`σ`]`:

i) There are no transitions from any state with boolean component 0 to a state with boolean component 1;

ii) The σ-transitions have all their target states in $\{(\top,0),(\top,1),(\top',1)\}$; and for any $\gamma \neq \sigma$, the target states of γ-transitions are all in $\{(\eta,0),(\eta,1),(s,1)\}$.

Theorem 1 *Let Q be any basic Core XPath query, t any given trdag, and let $\mathcal{G}$ denote either the rlag $\mathcal{D}_t$, or any given chibling $\mathcal{F}$ of $\mathcal{L}_t$. Assume given a labeling function Ł from $Nodes(\mathcal{G})$ into the set of ll-pairs, which is correct with respect to Q, i.e., in conformity with the semantics of subsection* 4.1. *Then there is a run r of the automaton A_Q on $\mathcal{G}$, such that :*

i) r is compatible with Ł; i.e., $r(u) = Ł(u)$ for every node u on $\mathcal{G}$;

*ii) r satisfies the maximal priority condition (***MP***) of subsection* 4.1.

Proof. We first construct, by induction, a 'complete' run (i.e., defined at all the nodes of $\mathcal{G}$) satisfying property i). For that, we shall employ reasonings that will be specific to the axis of the basic query Q. We give here the details only for the axis `parent`; they are similar for the other axes.

$Q =$ `//*[parent::`σ`]`: (The axis considered is non-sibling so $\mathcal{G} = \mathcal{D}_t$ here.) At the root u node of $\mathcal{D}_t$, we set $r(u) = Ł(u)$; we have to show that there is a transition rule in $\mathbf{A}_Q$ of the form $(init, llab(u)) \rightarrow Ł(u)$. Obviously, for the axis `parent`, the root node u cannot correspond to a node on t selected by Q, so the only ll-pairs possible for $Ł(u)$ are $(l,0),(l,1)$, with $l \in \{\eta,\top\}$; for each of these choices, we do have a transition rule of the needed form, on $\mathbf{A}_Q$.

Consider then a node v on $\mathcal{D}_t$ such that, at each of its ancestor nodes u on $\mathcal{D}_t$, the part of the run r of A_Q has been constructed such that $r(u) = Ł(u)$; assume that the run cannot be extended at the node by setting $r(v) = Ł(v)$. This means that there exists a parent node w of v, such that $(Ł(w), llab(v)) \rightarrow Ł(v)$ is not a transition rule of $\mathbf{A}_Q$; we shall then derive a contradiction. We only have to consider the cases where the boolean component of $Ł(w)$ is greater than or equal to that of $Ł(v)$. The possible couples $Ł(w), Ł(v)$ are then respectively:

$$Ł(w) : (\top,0) \mid (\top,1) \mid (\top,1) \mid (\top',1) \mid (\top',1)$$
$$Ł(v) : \ (\eta,0) \mid (\top,1) \mid \ (\eta,1) \mid (\top,1) \ \mid \ (\eta,1)$$

In all cases, we have $llab(w) = \sigma$ because of the semantics, so the node (on t corresponding to the node) v has a σ-parent, so must be selected; thus the above choices for $Ł(v)$ are not in conformity with the semantics; contradiction.

We now prove that the complete run r thus constructed, satisfies property ii). For this part of the proof, the reasoning does not need to be specific for each Q; so, write Q more generally, as `//*[axis::`σ`]` for some given σ. Suppose the run r does not satisfy the maximal priority condition at some node v on $\mathcal{G}$; assume, for instance, that the run r made the choice, say of the ll-pair $(l,1)$, although the maximal labeling of the node v, in a manner compatible with the ll-pairs of all its parents, was the ll-pair $(l,0)$. Since Ł is assumed correct, and r is compatible with Ł, the maximal possible labeling $(l,0)$ would mean that the node (on t corresponding to the node) v has no descendant selected by Q; whereas, the choice that r is assumed to have made at v, namely the ll-pair $(l,1)$, has the opposite semantics whether or not $llab(v) = \sigma$; in other words, the labeling Ł would not be correct with respect to Q; contradiction. The other possibilities for the 'bad' labelings under r also get eliminated in a similar manner. □

Theorem 2 *Let $Q, t, \mathcal{D}_t, \mathcal{F}, \mathcal{G}$ be as above. Let r be a (complete) run of the automaton A_Q on $\mathcal{G}$, which satisfies the maximal priority condition (***MP***) of subsection* 4.1. *Then the labeling function Ł on $Nodes(\mathcal{G})$, defined as $Ł(u) = r(u)$ for any node u, is correct with respect to the semantics of subsection* 4.1.

Proof. Let us suppose that the labeling Ł deduced from r is *not* correct with respect to Q; we shall then derive a contradiction. The reasoning will be by case analysis, which will be specific to the axis of the basic query Q considered. We give the details here for $Q =$ `//*[descendant::`σ`]`. The axis is non-sibling, so we have $\mathcal{G} = \mathcal{D}_t$ here. The sets $Nodes(t), Nodes(\mathcal{D}_t)$ are in a natural bijection, so for any node u on $\mathcal{D}_t$ we shall also denote by u the corresponding node on t, in our reasonings below.

We saw that the automaton $\mathbf{A}_Q$ for the `descendant` axis does not have the states $(\eta, 1), (\top, 1)$. Consider then a node u on $\mathcal{D}_t$ such that: for all ancestor nodes w of u, the llabel $r(w)$ is in conformity with the semantics, but the ll-pair $r(u)$ is not in conformity. Now, $\mathbf{A}_Q$ has only 5 states: $(init), (\top', 1), (s, 1), (\top, 0), (\eta, 0)$, of which only the last four can llabel the nodes. So the possible 'bad' choices that r is assumed to have made at our node u, are as follows:

(a) $r(u) = (\top', 1)$, but the node u is *not* an answer to the query Q. Here $name(u)$ must be σ, so the choice of r ought to have been $(\top, 0)$;

(b) $r(u) = (s, 1)$, but the node u is *not* an answer to the query Q. Here $name(u) \neq \sigma$, so the choice of r ought to have been $(\eta, 0)$;

(c) $r(u) = (\eta, 0)$, but the node u *is* an answer to the query Q. Here $name(u) \neq \sigma$, so the choice of r ought to have been $(s, 1)$;

(d) $r(u) = (\top, 0)$, but the node u *is* an answer to the query Q. Here $name(u)$ must be σ, so the choice of r ought to have been $(\top', 1)$.

In all the four cases, we have to show:

i) that the "ought-to-have-been" choice ll-pair is reachable from *all* the parent nodes of u;

ii) *and* that, with such a new and 'correct' choice made at u, r can be completed from u, into a run on the entire dag $\mathcal{D}_t$.

The reasoning will be similar for cases (a), (b), and for the cases (c), (d). Here are the details for case (a): That u is *not* an answer to Q means that u has *no* σ-descendant node, so for all nodes v below u on $\mathcal{D}_t$, we have $llab(v) \neq \sigma$. Therefore, assertions i) and ii) above follow from the following observations on the automaton for $Q =$ `//*[descendant::`σ`]`:

i) *if* r could reach the state $(\top', 1)$ at node u (via a σ-transition) from any parent node of u, then $(\top, 0)$ is also reachable thus at u, from any of them;

ii) *if*, from the state $(\top', 1)$, r could reach all the nodes on $\mathcal{D}_t$ below u (with state $(\eta, 0)$), via transitions over $\gamma \neq \sigma$, then it can do exactly the same now, with the 'correct' choice ll-pair $(\top, 0)$ at u.

As for case (c): Node u *is* an answer to Q here, so u has a σ-descendant; let v be a σ-node below u on $\mathcal{D}_t$; the ll-pair $r(v)$ that r assigns to v must then be either $(\top', 1)$ or $(\top, 0)$; this implies that r passed from the state $(\eta, 0)$ – supposedly assigned by r to u – to $(\top', 1)$ or $(\top, 0)$ somewhere between u and v; which is impossible, as is easily seen on the automaton $\mathbf{A}_Q$ for the axis `descendant`

considered. The reasoning for case (d) is even easier: from state $(\top, 0)$, no state with an outgoing σ-transition is reachable. □

6 Evaluating Composite Queries

Given a trdag t, we consider now queries which are disjunctive or conjunctive, of the form: $Q =$ `//*[axis`$_1$`::x]` ***conn*** `//*[axis`$_2$`::y]`, where $conn \in \{and, or\}$. To show how our approach is applied for evaluating such a query, we may assume wlog that $\texttt{axis}_1$ and $\texttt{axis}_2$ are basic. We apply the method described earlier, separately for $Q_1 =$ `//*[axis`$_1$`::x]` and for $Q_2 =$ `//*[axis`$_2$`::y]`, thus getting two respective evaluating runs r_1, r_2. Any node u of the dag $\mathcal{D}_t$ will then be relabeled, by the composite query Q, with ll-pairs computed by a function AND when $conn = and$ (resp. OR when $conn = or$), in conformity with the semantics presented in the subsection 4.1:

$AND(u) = (s, 1)$ iff $r_1(u) = (l', 1) = r_2(u)$;
$AND(u) = (\eta, 0)$ iff $r_1(u) = (l, 0)$ or $r_2(u) = (l, 0)$;
$AND(u) = (\eta, 1)$ otherwise.
$OR(u) = (s, 1)$ iff $r_1(u) = (l', 1)$ or $r_2(u) = (l', 1)$;
$OR(u) = (\eta, 0)$ iff $r_1(u) = (l, 0) = r_2(u)$;
$OR(u) = (\eta, 1)$ otherwise.

Figure 4 illustrates the above reasoning, for the evaluation of the composite query $Q =$ `//*[self::`b`]` ***and*** `//*[parent::`a`]`, on the trdag t of Figure 2:

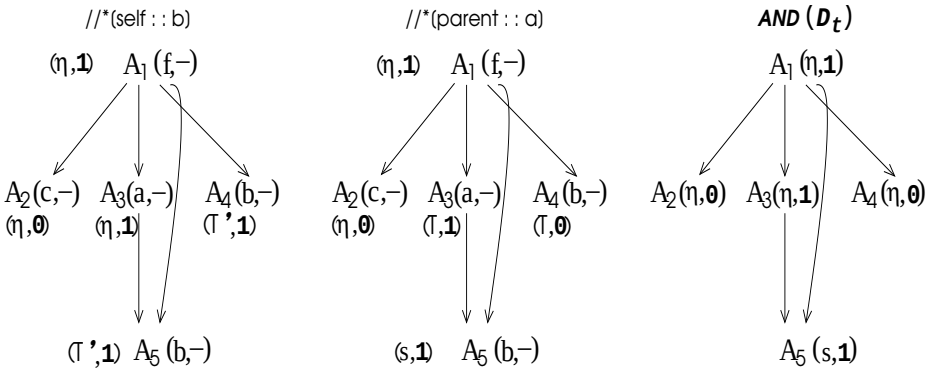


Fig. 4.

We next consider queries of the form $Q =$ `//*[axis_1::*[//*[axis_2::`σ`]]]`, that are imbricated; we again assume wlog that the two axes are basic. For their evaluation, we first consider a maximal priority evaluating run r_2 (resp. a set of runs $\widehat{r}_2$) of the automaton associated to the inner query `//*[axis_2::`σ`]`, on $\mathcal{D}_t$ (resp. the set of all chiblings of $\mathcal{L}_t$). This run (resp. set of runs) will output the rlag $r_2(\mathcal{D}_t)$ (resp. $\widehat{r}_2(\mathcal{D}_t)$), as described in subsection 4.2. Evaluating the imbricated query Q on the dag t is then done by running the automaton for the (basic) outer query `//*[axis`$_1$`::`s`]` on $r_2(\mathcal{D}_t)$ (resp. $\widehat{r}_2(\mathcal{D}_t)$).

7 Conclusion

Information retrieval from compressed structures, without having to uncompress them, is a field of active research, cf. e.g., [14, 9]. Our concern in this paper has been the evaluation of queries on XML documents that may be in a compressed form. Limiting our concern to positive Core XPath queries, we have presented a method for evaluating them on any trdag t without having to uncompress t: we first break up the given query into several sub-queries of a basic type; with each basic query, an automaton is associated such that a single unambiguous run of this automaton can evaluate the query. Such single runs of the automata are determined effectively by semantic labels that get attached to each node that the runs traverse. A direction for possible future work would be to adapt the approach presented here for evaluating more general XPath queries, such as those involving the namespace, or comprising negated sub-queries.

References

1. P. Buneman, M. Grohe, C. Koch, *Path queries on compressed XML*. In Proc. of the 29*th* Conf. on VLDB, 2003, pp. 141–152, Ed. Morgan Kaufmann.
2. G. Busatto, M. Lohrey, S. Maneth, *Grammar-Based Tree Compression*. EPFL Technical Report IC/2004/80, `http://icwww.epfl.ch/publications`.
3. G. Busatto, M. Lohrey, S. Maneth, *Efficient Memory Representation of XML Documents*. In Proc. DBPL'05 (to appear), LNCS 3774, Springer-Verlag, 2005.
4. B. Fila, S. Anantharaman, *Automata for Analyzing and Querying Compressed Documents*, Research Report, RR-2006-03, LIFO, March 2006, `http://www.univ-orleans.fr/lifo/prodsci/rapports/`
5. M. Frick, M. Grohe, C. Koch, *Query Evaluation of Compressed Trees*, In Proc. of LICS'03, IEEE, pp. 188–197.
6. G. Gottlob, C. Koch, *Monadic Queries over Tree-Structured Data*, In Proc. of LICS'02, IEEE,
7. G. Gottlob, C. Koch, *Monadic Datalog and the Expressive Power of Languages for Web Information Extraction*, In Journal of the ACM, 51(1):12–28, 2004.
8. G. Gottlob, C. Koch, R. Pichler, L. Segoufin, *The complexity of XPath query evaluation and XML typing* In Journal of the ACM 52(2):284-335, 2005.
9. M. Lohrey, *Word problems and membership problems on compressed words* In SIAM Journal of Computing, 35(5):1210-1240, 2006.
10. M. Marx, *XPath and Modal Logics for Finite DAGs*. In Proc. of TABLEAUX'03, pp. 150–164, LNAI 2796, 2003.
11. W. Martens, F. Neven, *On the complexity of typechecking top-down XML transformations*, In Theoretical Computer Sc., 336(1): 153–180, 2005.
12. F. Neven, *Automata Theory for XML Researchers*, In SIGMOD Record 31(3), September 2002.
13. F. Neven, T. Schwentick, *Query automata over finite trees*, In Theoretical Computer Science, 275(1–2):633–674, 2002.
14. W. Rytter, *Compressed and fully compressed pattern matching in one and two dimensions*, In Proceedings of the IEEE, 88(11):1769–1778, 2000.
15. Worl Wide Web Consortium, *XML Path Language (XPath Recommendation)*, `http://www.w3c.org/TR/xpath/`

Appendix: Constructing the Maximal Priority Run

Given a trdag t, we give here the algorithm constructing the maximal priority run of the automaton $\mathbf{A}_Q$ for any basic non-sibling query Q. (It is easily adapted for the sibling queries by replacing $\mathcal{D}_t$ by the chiblings of $\mathcal{L}_t$.) We set n :=number of nodes of $\mathcal{D}_t$. The nodes on $\mathcal{D}_t$ are referred to by their names: $A_i, 1 \leq i \leq n$. Recall that Δ_Q is the set of transition rules of $\mathbf{A}_Q$.

INPUT: The rlag $\mathcal{D}_t$; and the automaton $\mathbf{A}_Q$;
BEGIN :
$next(\eta, 0) := (\eta, 1)$; $next(\eta, 1) := (s, 1)$; $next(s, 1) := \emptyset$;
$next(\top, 0) := (\top, 1)$; $next(\top, 1) := (\top', 1)$; $next(\top', 1) := \emptyset$;
$r(A_0) := init$; $\quad Sons(A_0) := \{A_1\}$;
For all $j \in \{1, \ldots, n\}$
If $llab(A_j) \neq \sigma$, $E_j := [(\eta, 0), (\eta, 1), (s, 1)]$ /* See as lists */
Else $E_j := [(\top, 0), (\top, 1), (\top', 1)]$;
$i := 1$;
(1) $r(A_i) := first(E_i)$; /* Choose the first ll-pair in E_i */
If for all A_j with $A_i \in Sons(Aj)$, $(r(A_j), llab(A_i))) \rightarrow r(A_i) \in \Delta_Q$
$i := i + 1$;
If $i \leq n$ **goto** (1)
Else return $r(A_k)$ for all $k \in \{1, \ldots, n\}$;
/* Case of no selection: each A_i gets ll-pair $(l, 0)$, corresp. to its llabel */
Else /* Now case with selection*/
$E_1 := E_1 \setminus [(\eta, 0), (\top, 0)]$; /* First element in E_1 is removed */
$i := 1$;
(2) $r(A_i) := first(E_i)$; /* Choose the first ll-pair in E_i */
(3) **If** for all A_j with $A_i \in Sons(Aj)$, $(r(A_j), llab(A_i))) \rightarrow r(A_i) \in \Delta_Q$
$i := i + 1$;
If $i \leq n$ **goto** (2)
Else return $r(A_k)$ for all $k \in \{1, \ldots, n\}$;
Else
(4) **If** $next(r(A_i)) \neq \emptyset$ {
$r(A_i) := next(r(A_i))$;
goto (3); /* Try with next ll-pair at A_i */
}
Else $\{i := i - 1$; **goto** (4)$\}$; /* Try with next ll-pair at A_{i-1} */
END.

Remark 3; The above algorithm computes the word of length n over the alphabet formed of ll-pairs, which is maximal for the descending lexicographic order defined by the priority relation $>$. At each position i on the word (corresponding to a node on $\mathcal{D}_t$), 3 ll-pairs are possible (determined by the symbol at that node on $\mathcal{D}_t$). The worst case complexity of the algorithm is $O(n^6)$: there are n^3 possible words, and the construction of each costs at most n^3 steps.