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. Given any dag t representing a fully or partially compressed XML document, we present a method for evaluating any positive unary query expressed in terms of Core XPath axes, on t, without unfolding t into a tree. 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 straightline regular tree grammar associated to the given dag, 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, on the dag t, can then be expressed as a sub-dag 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, 11, 12]); on the other hand, the notion of a compressed XML document has been introduced in [1, 5, 9], and a possible advantage of using dag structures for the manipulation of such documents has been brought out in [9]. 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 any XML document possibly given in a compressed format. With such an objective, we first define the notion of a compressed document as a *tree/dag* (*trdag*, for short), designating a directed acyclic graph that may be partially or fully compressed; the terms 'trdag' and 'document' will therefore be considered synonymous in the sequel. We adopt then the view that a trdag t is equivalent to a minimal 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 construct 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 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 (trdag) 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, though not deterministic, 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 Q can be evaluated on t by combining the answers to its various basic sub-queries, and the answer set for Q is expressed as a sub-trdag of t, whose nodes get labeled in conformity with the semantics. It is important to note that the evaluation is performed on the *given* trdag t; as such, on two different trdags corresponding to two different compressions of the same XML tree, the answers obtained may *not* be the same, in general.

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 $\mathcal{D}_t$ 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.

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 $name_t(u)$ or $name(u)$;*
- *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, and denoted as $t|_u$. 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 any node u on 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, $pos_t(u) = \{\alpha.i \mid \alpha \in pos_t(v), v \text{ is a parent of } u, u \text{ is the i-th child of } v\}$. The elements of $Pos(t)$ are words over natural integers.

The function $name_t$ is extended naturally to the positions in $Pos(t)$ as follows: for every $u \in Nodes(t)$ and $\alpha \in pos_t(u)$, we set $name_t(\alpha) = name_t(u)$. Given a trdag t, we define its tree-equivalent as a tree $\hat{t}$ such that: $Pos(\hat{t}) = Pos(t)$, and for every $\alpha \in Pos(t)$ we have $name_{\hat{t}}(\alpha) = name_t(\alpha)$. A trdag is said to be a *tdag*, or *fully compressed*, iff for any two distinct nodes u, u' on t, the two sub-dags $t|_u$ and $t|_{u'}$ have non-isomorphic tree-equivalents; otherwise, the trdag is said to be *partially compressed*. For example, the tree to the left of Figure 1 is the tree-equivalent of the partially compressed trdag to the right, and also of the fully compressed tdag to the middle.

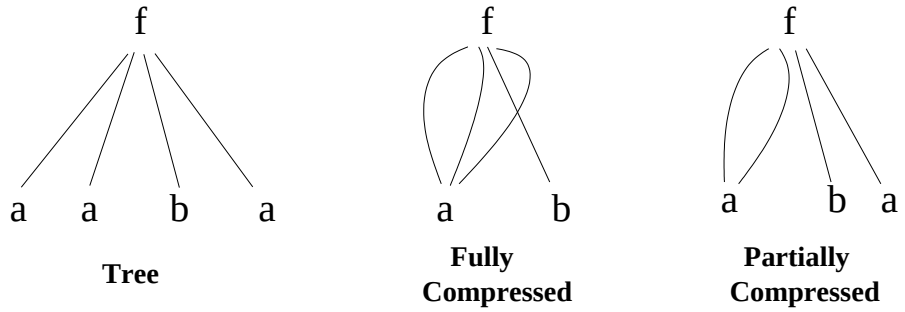


Fig. 1. tree, tdag, and trdag

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 of $\mathcal{L}_t$ 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.

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 $\mathcal{L}_t$.

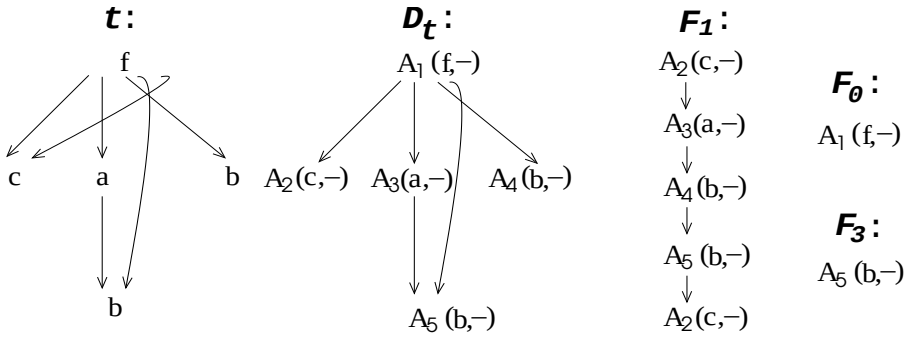


Fig. 2. trdag t, associated rlag $\mathcal{D}_t$, and chiblings of $\mathcal{L}_t$

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 is referred to as 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 Core XPath is the navigational segment of XPath, and is based on the following axes of XPath (cf. [7, 14]): `self, child, parent, ancestor, descendant, following-sibling, preceding-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. [14]). 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 2 *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....*

For the 'composite' axes `descendant-or-self` and `ancestor-or-self`, the semantics are then deduced in an obvious manner. We shall also need position predicates of the form `[`*position*`()=`i`]`; their semantics is that the expression `[child::`b `[`*position*`()=` i`]]` evaluates to *true* at a context node u, iff: `[child::`b`]` evaluates to *true* at u, and u is an i-th child of some parent.

Positive Core XPath query expressions are usually defined in the literature (cf. e.g., [5]), as those generated by the following grammar:

`A` ::= `self` | `child` | `descendant` | `parent` | `ancestor` | `preceding-sibling` | `following-sibling`
S_{can} ::= `A::`σ | *position*`()=`i | S_{can} ***and*** S_{can} | S_{can} ***or*** S_{can}
E_{can} ::= `A::*[`S_{can}`]` | E_{can}`[`E_{can}`]`
Q_{can} ::= `/`S_{can} | `/`E_{can} | Q_{can}`/`Q_{can}

We shall refer to the query expressions generated by this grammar as *canonical*; they can be shown to be of the type $/C_1/C_2/\ldots/C_n$, where each C_i is of the form `A::`σ`[`X_{can}`]`, or of the form `A::`σ`[`X_{can}`]` ***conn*** `A'::`σ'`[`X'_{can}`]`, with ***conn*** $\in \{and, or\}$, and $X_{can}, X'_{can} \in \{S_{can}, E_{can}, true\}$; we agree here to identify `A::`σ`[`*true*`]` with `A::`σ.

Any such positive Core XPath query expression can be translated into one that is in "standard form", i.e., where the format of the sub-queries is of the type '`axis::`b'; we formalize this idea now. We shall refer to the axes `self, child, descendant, parent, ancestor, preceding-sibling, following-sibling` as *basic*. A basic Core XPath query is 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_{std} generated by the following grammar, where σ stands for any node name on the documents, or for $*$ (meaning 'any'):

`A` ::= `self` | `child` | `descendant` | `parent` | `ancestor` | `preceding-sibling` | `following-sibling`
S ::= `A::`σ | *position*`()=`i | S ***and*** S | S ***or*** S | `Root`
E ::= `A::*[`S`]` | E`[`E`]`
Q_{std} ::= `//*` | `//*[`S`]` | `//*[`E`]`

Core XPath queries Q_{std} of the format generated by this grammar are said to be in *standard form*; to be able to handle any positive Core XPath query with such a grammar, we have introduced a special predicate called `Root`, deemed true only at the root node of the trdag considered.

By the *evaluation* of a given query expression Q on any trdag t, we mean the assignment: $t \mapsto$ the set of all context nodes on t where the expression Q evaluates to true (following the conventions of Definition 2); this latter set is also called the *answer* for Q on t. Two given queries Q_1, Q_2 are said to be *equivalent* iff, on any trdag t, the answer sets for Q_1 and Q_2 are the same. Any positive Core XPath query Q_{can} can be translated into an equivalent one in standard form; e.g., `/`c`[following-sibling::`g`]/`d is equivalent to `//*[self::`d *and* `parent::*[Root` *and* `self::`c `[following-sibling::`g`]]]` in standard form. An inductive procedure performing such a translation in the general case (of linear complexity w.r.t. the number of location steps in Q_{can}) is given in the Appendix. The following proposition results from Definition 2.

Proposition 1 *(1) For any set of nodes X on a trdag t, and any axis* `A`*, we have:* $\texttt{A}(X) =$

$$\bigcup_{\substack{x\in X,\ \alpha\in pos_t(x)\\ \alpha=i_1\ldots i_k}} \{\texttt{/child::*[}position\texttt{()=}i_1\texttt{]/.../child::*[}position\texttt{()=}i_k\texttt{]/A::*}\}$$

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

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

$$\bigcup_u \{\texttt{descendant-or-self(following-sibling(}\ \texttt{//*[self::}u\ \textit{and}\ \texttt{(descendant::}b\ \textit{or}\ \texttt{self::}b\texttt{)]}\ \texttt{))}\}$$

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

$$\bigcup_u \{\texttt{descendant-or-self(preceding-sibling(}\ \texttt{//*[self::}u\ \textit{and}\ \texttt{(descendant::}b\ \textit{or}\ \texttt{self::}b\texttt{)]}\ \texttt{))}\}$$

Finally, following [1], for any set S of nodes on t, the sets of nodes `following(S)` and `preceding(S)` can now be defined formally, as follows:

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

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

Note: Unlike on a tree, the `ancestor, descendant, following, self` and `preceding` axes do *not* partition the set of nodes on a trdag t, in general.

4 Automata for Basic Core XPath 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 first component ℓ will have 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 $(q, \tau) \rightarrow q'$ 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 $label(u)$. 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 automaton $\mathbf{A}_Q$ and from the states it attaches to the nodes of $\mathcal{D}_t$, we will deduce, at every node u of t, a well-determined ll-pair as (a new) label at u, via the natural bijection between $Nodes(t)$ and $Nodes(\mathcal{D}_t)$. The ll-pairs thus attached to the nodes of t will have the following semantics (where x stands for the name of the node u on t, corresponding to the 'current' node on $\mathcal{D}_t$):

- $(s,1)$: $x \neq \sigma$, 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;
- $(\top',1)$: $x = \sigma$, 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.

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 automaton $\mathbf{A}_Q$ 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 which are unique (or unambiguous in a sense that will be presently made clear), we define the following *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 $\mathcal{G}$ with an ll-pair coming either from the group $\{(\eta,0),(\eta,1),(s,1)\}$, or from the group $\{(\top,0),(\top,1),(\top',1)\}$; 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 $(\natural, x)$ 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 $(r(u), llab(v)) \to (\natural, x)$, 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 $\mathcal{D}_t$; 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, to any node of $\mathcal{D}_t$, named with a non-terminal A, will correspond in general a *set* of ll-pairs, assigned by the various runs of $\mathbf{A}_Q$ to the A-named nodes on the various chiblings of $\mathcal{L}_t$. We therefore proceed as follows: for every complete set $\widehat{r}$ of runs of $\mathbf{A}_Q$, formed of one run $r_{\mathcal{F}}$ 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 associate to $\widehat{r}$ and any $u \in Nodes(D_t)$, a set of ll-pairs:

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

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:

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

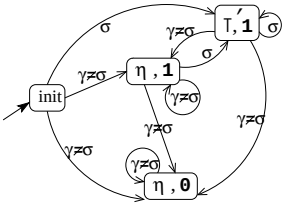
From $\mathcal{D}_t$ and this function $\lambda_{\widehat{r}}$, we next derive an rlag $\lambda_{\widehat{r}}(\mathcal{D}_t)$ by re-labeling each node u on $\mathcal{D}_t$ with the pair $(\lambda_{\widehat{r}}(u), -)$. And finally we define $\widehat{r}(\mathcal{D}_t)$ as the rlag obtained from $\lambda_{\widehat{r}}(\mathcal{D}_t)$, 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 llab s, and 0 otherwise. All these details are illustrated with an example in the following subsection.

4.3 The Automata

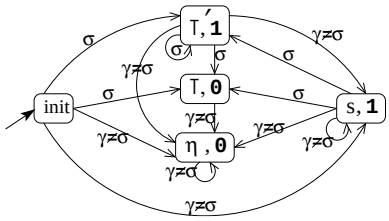
We present the automata for all the basic queries. A few words on some of the automata by way of explanation. First, the reason why the automaton for `self` does *not* have the states $(\top,0),(\top,1),(s,1)$: for $(\top,0),(\top,1)$, by the semantics of Section 4.1 we must have $x = \sigma$, where 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.

Automata for:

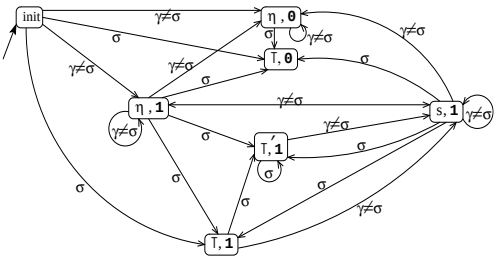
- //*[self::σ]



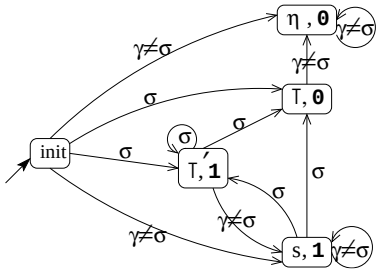
- //*[descendant::σ]



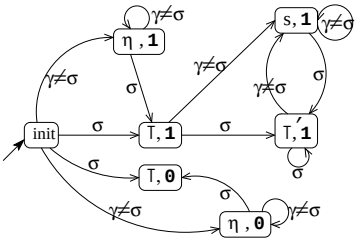
- //*[parent::σ]



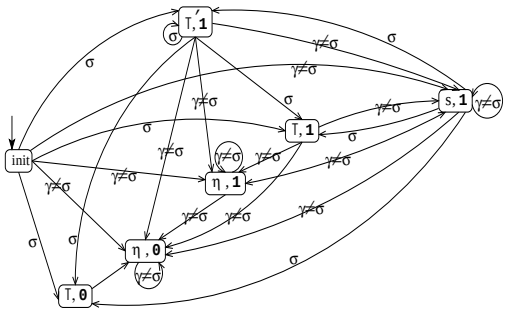
- //*[following-sibling::σ]



- //*[preceding-sibling::σ]



- //*[child::σ]



- //*[ancestor::σ]

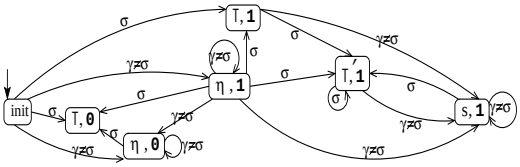


Figure 3 below illustrates the evaluation of $Q =$ `//*[following-sibling::`b`]`, on the trdag t of Figure 2. We first use the automaton for the basic query `//*[following-sibling::`σ`]` with $\sigma = b$, and then the automaton for `//*[self::`σ`]` with $\sigma = s$. The sub-trdag of t, formed of nodes corresponding to those of $\widehat{r}(\mathcal{D}_t)$ with labels having boolean component 1, contains all the answers to Q on t.

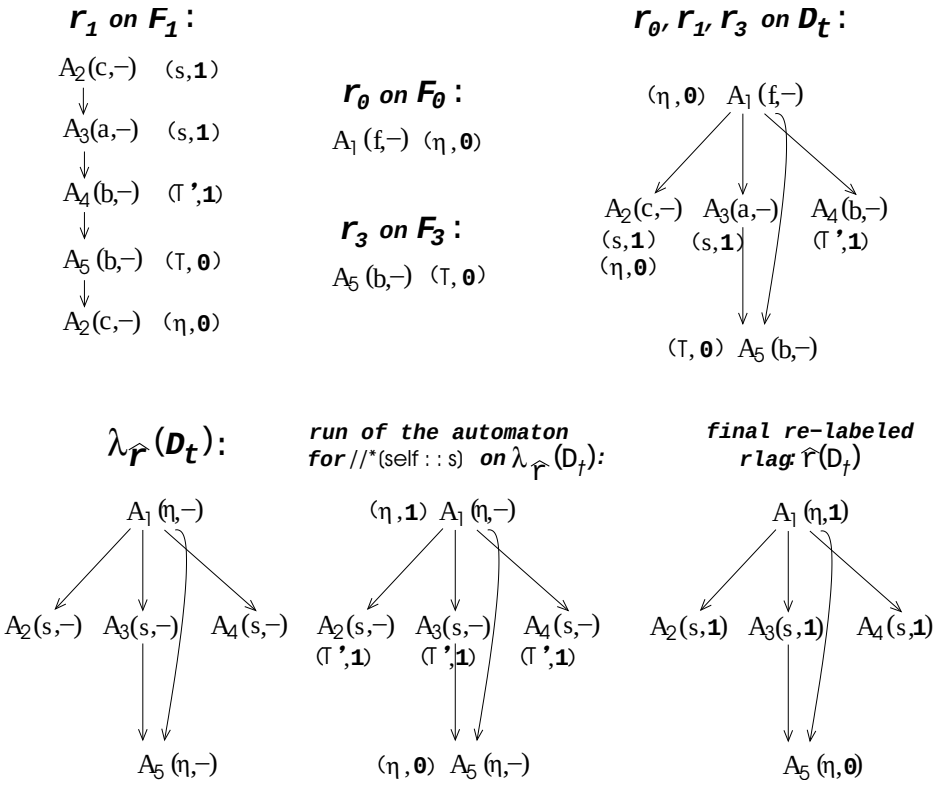


Fig. 3.

5 Maximal Priority Runs of Basic Query Automata

Note that the following properties, required by our semantics of Section 4.1, hold on the automaton $\mathbf{A}_Q$ 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 Section* 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 Section* 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*

Section 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 Section* 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

A composite query is a query in standard form, but not basic; it is evaluated incrementally. We first consider queries of the form `//*[A::`x ***conn*** `A'::`x'`]`, where ***conn*** $\in \{and, or\}$, where the axes are all basic. Observe that the answer for $Q =$ `//*[A::`x ***conn*** `A'::`x'`]` can be obtained as union (resp. intersection) of the answers for the two 'component' queries `//*[A::`x`]`, and `//*[A'::`x'`]`, when ***conn*** is an *or* (resp. *and*). So, we apply the method described earlier, separately for $Q_1 =$ `//*[A::`x`]` and for $Q_2 =$ `//*[A'::`x'`]`, thus getting two respective evaluating runs r_1, r_2. Any node u of the dag $\mathcal{D}_t$ will then be re-labeled, 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 of Section 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:

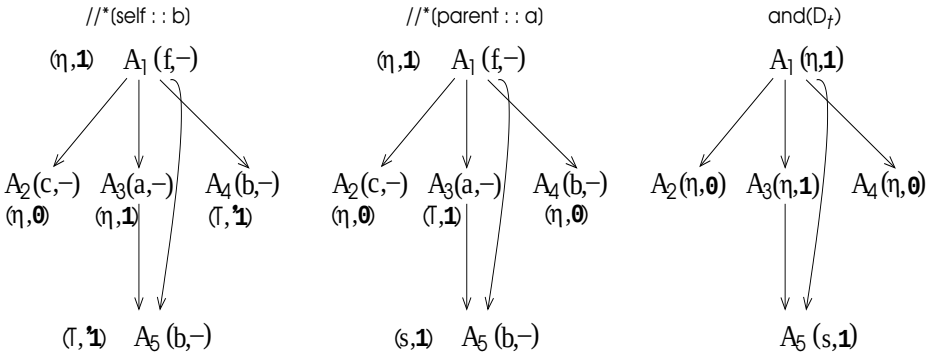


Fig. 4.

Next, we consider imbricated queries of the form $Q =$ `//*[`A_1`::*[`A_2`::`σ`]]`. We first consider a maximal priority run evaluating r_2 (resp. a set of runs $\widehat{r}_2$) of the automaton associated to the inner query `//*[`A_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 Section 4.2. Evaluating the given imbricated query Q on the dag t is then done by running the automaton for the basic outer query `//*[`A_1`::`s`]` on $r_2(\mathcal{D}_t)$ (resp. $\widehat{r}_2(\mathcal{D}_t)$).

Finally, the answer for a query of the type $Q =$ `//*[child::`x`[`*position*`()=`k`]]`, is the subset of the nodes answering `//*[child::`x`]`, which correspond to a k-th node on some chibling.

7 Conclusion

Information retrieval from compressed structures, without having to uncompress them, is a field of active research; cf. e.g., [13, 8]. Our concern 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, by breaking up the given query –in linear time, w.r.t. the number of location steps– into sub-queries of a basic type; with each basic query, an automaton is associated such that an unambiguous maximal priority run of this automaton can evaluate the query. An algorithm constructing the maximal priority runs is given in [4]; it has just been implemented. It is of complexity $\mathcal{O}(n^3)$ on any trdag t, where n is the number of nodes of t; it reduces to $\mathcal{O}(n^2)$ on trees, where the relation $Parents$ is trivial. (*Please note*: given a query Q, the answer for Q on a trdag t can be a strict superset of the answer for the same query Q on the tree-equivalent $\hat{t}$ of t; it is shown in [4] how to derive the answer for Q on $\hat{t}$, from the answer for Q on t; the method consists in completing the label associated to any given node u on t by the run of $\mathbf{A}_Q$, by an appropriate subset of $pos_t(u)$.)

An advantage of the approach presented in this paper seems to be that the basic sub-queries "composing" a given query can be evaluated in parallel, in several cases; a detailed analysis of this issue could be a direction for future work. We also expect to be able to extend our approach to the evaluation of more general XPath queries, such as those involving the data values, by adapting its underlying mechanism based on labeling.

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, LNCS 3774, pp. 199–216, Springer-Verlag, 2005.
4. B. Fila, S. Anantharaman, *Automata for Analyzing and Querying Compressed Documents*, Research Report, RR-2006-03, LIFO, 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, pp. 188–197, IEEE, 2003.
6. G. Gottlob, C. Koch, *Monadic Queries over Tree-Structured Data*, In Proc. of LICS'02, pp. 189–202, IEEE, 2002.
7. 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.
8. M. Lohrey, *Word problems and membership problems on compressed words* In SIAM Journal of Computing, 35(5):1210-1240, 2006.
9. M. Marx, *XPath and Modal Logics for Finite DAGs.* In Proc. of TABLEAUX'03, pp. 150–164, LNAI 2796, 2003.
10. W. Martens, F. Neven, *On the complexity of typechecking top-down XML transformations*, In Theoretical Computer Sc., 336(1): 153–180, 2005.

11. F. Neven, *Automata Theory for XML Researchers*, In SIGMOD Record 31(3), September 2002.
12. F. Neven, T. Schwentick, *Query automata over finite trees*, In Theoretical Computer Science, 275(1–2):633–674, 2002.
13. W. Rytter, *Compressed and fully compressed pattern matching in one and two dimensions*, In Proceedings of the IEEE, 88(11):1769–1778, 2000.
14. Worl Wide Web Consortium, *XML Path Language (XPath Recommendation)*, `http://www.w3c.org/TR/xpath/`

Appendix: From Canonical Forms to Standard Forms

We stick to the notations of Section 3.1. Given any canonical Core XPath query expression Q_{can}, we compute, inductively, an equivalent standard XPath expression denoted as $Std(Q_{can})$; as earlier, **conn** stands for either of the boolean connectives *and*, *or*.

To start with, we define:

$Std(\texttt{[}true\texttt{]}) = \texttt{self::}*$
$Std(\texttt{[}S_{can}\texttt{]}) = S_{can}$
$Std(\texttt{[A::}\sigma\texttt{[}S_{can}\texttt{]]}) = \texttt{A::}*\texttt{[self::}\sigma\ and\ Std(\texttt{[}S_{can}\texttt{]})\texttt{]}$
$Std(\texttt{[A::}\sigma\texttt{[A}_1\texttt{::}\sigma_1\texttt{[...[A}_k\texttt{::}\sigma_k\texttt{]...]]]}) = \texttt{A::}*\texttt{[self::}\sigma\ and$
$\texttt{A}_1\texttt{::}*\texttt{[self::}\sigma_1\ and \ldots\ \texttt{A}_{k-1}\texttt{::}*\texttt{[self::}\sigma_{k-1}\ and\ \texttt{A}_k\texttt{::}\sigma_k\texttt{]}\ \ldots\texttt{]]}$

We also define, for every basic axis relation, an inverse relation, as follows:

$\texttt{self}^{-1} = \texttt{self}$, $\texttt{child}^{-1} = \texttt{parent}$, $\texttt{parent}^{-1} = \texttt{child}$,
$\texttt{ancestor}^{-1} = \texttt{descendant}$, $\texttt{descendant}^{-1} = \texttt{ancestor}$
$\texttt{following-sibling}^{-1} = \texttt{preceding-sibling}$,
$\texttt{preceding-sibling}^{-1} = \texttt{following-sibling}$

For any query $Q = //*[X]$ in standard form, we set $exp(Q) = X$. For any canonical Core XPath query $Q = /C_1/C_2/\ldots/C_n$, the standard form $Std(Q)$ of Q is then generated by the following recursive construction:

Case of length 1: $Q = /C_1$
$Std(\texttt{/child::}\sigma\texttt{[}X_{can}\texttt{]}) = \texttt{//*[(Root}\ and\ \texttt{self::}\sigma\texttt{)}\ and\ Std(\texttt{[}X_{can}\texttt{]})\texttt{]}$
$Std(\texttt{/child::*[}X_{can}\texttt{]}) = \texttt{//*[Root}\ and\ Std(\texttt{[}X_{can}\texttt{]})\texttt{]}$
$Std(\texttt{/descendant::}\sigma\texttt{[}X_{can}\texttt{]}) = \texttt{//*[self::}\sigma\ and\ Std(\texttt{[}X_{can}\texttt{]})\texttt{]}$
$Std(\texttt{/descendant::*[}X_{can}\texttt{]}) = \texttt{//*[}Std(\texttt{[}X_{can}\texttt{]})\texttt{]}$
$Std(\texttt{/axis::}\sigma\texttt{[}X_{can}\texttt{]}\ \textbf{conn}\ \texttt{axis'::}\sigma'\texttt{[}X'_{can}\texttt{]} =$
$\texttt{//*[}exp(Std(\texttt{/axis::}\sigma\texttt{[}X_{can}\texttt{]}))\ \textbf{conn}\ exp(Std(\texttt{/axis'::}\sigma'\texttt{[}X'_{can}\texttt{]}))\texttt{]}$

Case of length n>1: $Q = /C_1/C_2/\ldots/C_n$
$Std(/C_1/\ldots/C_{n-1}\texttt{/A::}\sigma\texttt{[}X_{can}\texttt{]}) =$
$\texttt{//*[(self::}\sigma\ and\ Std(\texttt{[}X_{can}\texttt{]}))\ and\ \texttt{A}^{-1}\texttt{::*[}exp(Std(/C_1/\ldots/C_{n-1}))\texttt{]]}$
$Std(/C_1/\ldots/C_{n-1}\texttt{/A::}\sigma\texttt{[}X_{can}\texttt{]}\ \textbf{conn}\ \texttt{A'::}\sigma'\texttt{[}X'_{can}\texttt{]}) =$
$\texttt{//*[((self::}\sigma\ and\ Std(\texttt{[}X_{can}\texttt{]}))\ \textbf{conn}\ \texttt{(self::}\sigma'\ and\ Std(\texttt{[}X'_{can}\texttt{]})))$
$and\ \texttt{A}^{-1}\texttt{::*[}exp(Std(/C_1/\ldots/C_{n-1}))\texttt{]]}$

This translation procedure is of complexity linear with respect to the number of location steps (i.e. of the form `axis::`σ) that appear in Q.