UNIVERSITE D'ORLEANS
Faculté des Sciences
LIFO

Laboratoire d'Informatique Fondamentale d'Orléans
4, rue Léonard de Vinci, BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche
www : http://www.univ-orleans.fr/SCIENCES/LIFO/

Automata for Analyzing and Querying Compressed Documents

Barbara FILA, LIFO, Orléans (Fr.)
Siva ANANTHARAMAN, LIFO, Orléans (Fr.)

Rapport N^o **2006-03**

Automata for Analyzing and Querying Compressed Documents

Barbara Fila, Siva Anantharaman

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

Abstract. In a first part of this work, tree/dag automata are defined as extensions of (unranked) tree automata which can run indifferently on trees or dags; they can thus serve as tools for analyzing or querying any semi-structured document, whether or not given in a compressed format. In a second part of the work, we present a method for evaluating positive unary queries, expressed in terms of Core XPath axes, on any dag t representing an XML document possibly given in a compressed form; the evaluation is done directly on t, without unfolding it 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 minimal straightline regular tree grammar associated to the given dag t, or along complete sibling chains in this grammar. Any given positive 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 whose nodes are suitably labeled under the runs of such automata.

Keywords: Tree 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., [8, 9, 15, 16]); on the other hand, the notion of a compressed XML document has been introduced in [2, 7, 12], and a possible advantage of using dag structures for the manipulation of such documents has been brought out in [12]. It is legitimate then to investigate the possibility of using automata over dags instead of over trees, for querying compressed XML documents.

Dag automata (DA) were first introduced and studied in [5]; a DA was defined there as a natural extension of tree automaton, i.e. as a bottom-up tree automaton running on dags; and the language of a DA was defined as the set of dags that get accepted under (bottom-up) runs, defined in the usual sense; the emptiness problem for DAs was shown there to be NP-complete, and the membership problem proved to be in NP; but the problem of stability under complementation of the class of dag automata –closely linked with that of determinization– was left open. These two issues have since been settled negatively in [1]: the reason is that the set of all terms (trees) represented by the set of dags accepted by a *non-deterministic* DA is not necessarily a regular tree language; a consequence is that the class of tree languages recognized by DAs (as sets of accepted dags) is a *strict* superclass of the class of regular tree languages. It is well-known however, that answers to MSO-definable queries on (semi-)structured trees form regular tree languages ([18]); it is thus necessary to define the languages of DAs in a manner *different* from that of [5, 1], if they are to serve as tools for analyzing and querying a document, independently of whether it is given in a (partially or fully) compressed format, or as a tree. Our first aim in this work is therefore to redefine the notion of the language of a DA suitably, with such an objective.

For achieving that, we first present (in Section 2) 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 terminology *trdag* has been chosen to distinguish it from that of *tdag* employed in [1]; this latter term will be employed in this paper when referring to a fully compressed dag. A Tree/Dag automaton (TDA, for short) is then defined as an automaton which runs on trdags. The essential differences with the DAs of [1] are the following: (i) our TDAs can be unranked, and (ii) although the transition rules of a TDA look quite like those of the DAs in [1], or those of TAs, a run of a TDA on any given *trdag* t will carry with it not only assignments of states to the nodes of t, but also to the edges of t; runs will be so defined that a TDA accepts any given trdag t if and only if it accepts the tree $\hat{t}$ obtained by uncompressing t, as a tree automaton running on the tree $\hat{t}$, in the usual sense.

In the second part of the paper, we present an approach based on word automata for evaluating queries on trdags that represent XML documents in a partially or fully compressed format; the terms 'trdag' and 'document' will therefore be considered synonymous in the sequel. Any given trdag t is first seen as equivalent to a minimal straightline regular tree grammar $\mathcal{L}_t$, that one can naturally associate with t, cf. e.g., [3, 4]. 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 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 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. 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 a same XML tree, the answers obtained may *not* be the same, in general.

The paper is structured as follows: Section 2 presents the notions of trdags, and of Tree/Dag automata. 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. In Section 7, we show how to refine our approach, so as to derive, from the answer for any given Core Xpath query Q on a trdag t, the answer set for the same query Q on the tree-equivalent $\hat{t}$ of t. In the appendices, we show how to translate the 'usual' Core XPath queries into one in 'standard' form on which our approach is applicable (the translation is done in linear time on the size of the given query); we also present a polynomial time algorithm for constructing the maximal priority run, for any basic query automaton over any given document (trdag), with a complexity bound of $\mathcal{O}(n^3)$, where n is the number of nodes of the trdag; the bound reduces to $\mathcal{O}(n^2)$ on trees where the relation *Parents* is trivial; a complete illustrative example, on a composite imbricated query, is given in the last appendix.

2 Tree/Dag Automata

Definition 1 *A tree/dag (*trdag *for short) over a not necessarily ranked alphabet Σ is a rooted dag (directed acyclic graph) $t = (Nodes(t), Edges(t))$, where, for any node $u \in Nodes(t)$:*
- *u has a* name $name_t(u) = name(u) \in \Sigma$*;*
- *the edges going out of any node are ordered;*
- *and if $name(u)$ is ranked, then the number of outgoing edges at u is the rank of $name(u)$.*

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 every $1 \le i \le n$, 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 $v \xrightarrow{i} u_i$; the set of all outgoing (resp. incoming) edges at any node v will be denoted as $Out_v(t)$, or Out_v (resp. $In_v(t)$, or In_v); and for any node u, we set: $Parents(u) = \{v \in Nodes(t) \mid u \text{ is a child of } v\}$.

A trdag t will be 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 set $Pos(t)$ consists of (some of the) words over natural integers. To any edge $\mathbf{e} : u \xrightarrow{i} v$ on a trdag t, is naturally associated the subset $pos_t(\mathbf{e}) = pos_t(u).i$ of $Pos(t)$.

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 $name_t(\alpha) = name_{\hat{t}}(\alpha)$. It is immediate that $\hat{t}$ is uniquely determined, up to a tree isomorphism; it can actually be constructed canonically (cf. [7]), by taking for nodes the set $Pos(t)$, and for directed edges the set $\{(\alpha, \alpha.i) \mid \alpha, \alpha.i \in Pos(t)\}$, each node α being named with $name_t(\alpha)$. There is then a natural, name preserving, surjective map from $Nodes(\hat{t})$ onto $Nodes(t)$; it will be referred to in the sequel as the compression map, and denoted as $\mathbf{c}$.

A trdag is said to be a *tdag*, or *fully compressed*, iff for any two different 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* when it is not a tree. For example, the tree to the left of Figure 1 is the tree-equivalent of the partially compressed trdag to the right, and also to the fully compressed tdag to the middle.

We define now the notion of a Tree/Dag automaton, first over a ranked alphabet Σ, to facilitate understanding. The definition is then easily extended to the unranked case.

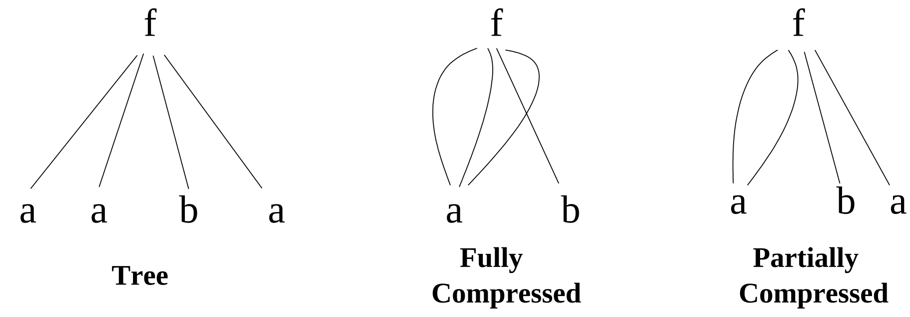


Fig. 1. tree, tdag, and trdag

Definition 2 *A Tree/Dag automaton (TDA, for short) over a ranked alphabet Σ is a tuple (Σ, Q, F, Δ), where Q is a finite non-empty set of states, $F \subseteq Q$ is the set of final (or accepting) states, and Δ is a set of transition rules of the form: $f(q_1, ..., q_k) \to q$, where $f \in \Sigma$ is of rank k, and $q_1, \dots, q_k, q \in Q$.*

It will be convenient to write the transition rules of a TDA in a different (but equivalent) form: a transition of the form $f(q_1, \dots, q_k) \to q$ is also written as $(f, q_1 \dots q_k) \to q$, where $q_1 \dots q_k$ is seen as a word in Q^*, of length $= rank(f)$ in the ranked case. The notion of a TDA is then extended easily to the unranked case, i.e., where the signature symbols naming the nodes are not assumed to be of fixed rank: it suffices to define the transitions to be of the form $(f, \omega) \to q$, where $\omega \in Q^*$; we may assume wlog that ω is a '$*$'-regular expression on Q not involving '$+$', by replacing a rule $(f, \omega + \omega') \to q$, by the two rules $(f, \omega) \to q, (f, \omega') \to q$.

A TDA is said to be bottom-up *deterministic* iff whenever there are two transition rules of the form $(f, \omega) \to q$, $(f, \omega') \to q'$, with $q \neq q'$, we have necessarily $\omega \cap \omega' = \emptyset$; otherwise it is said to be *non-deterministic*. We also agree to denote the transitions of the form $(f, \emptyset) \to q$ simply as $f \to q$, and refer to them as *initial* transitions.

For defining the notion of runs of TDAs on a trdag in a bottom-up style, we need some preliminaries. Let $\mathbf{A}$ be a TDA with state set Q and transition set Δ. Suppose t is a trdag and assume given a map $M : Edges(t) \to Q$. If u is any node on t with $u_1 \dots u_n$ as the string of all its (not necessarily distinct) children, the string $M(\mathbf{e}(u, 1)) \dots M(\mathbf{e}(u, n))$, formed of states assigned by M to the outgoing edges at u, will be denoted as $M(Out_u)$. We then define, recursively in a bottom-up style, a binary relation at u on the states of Q, with respect to (w.r.t. or wrt, for short) the given map M; this relation, denoted as $\lhd_u^M = \lhd_u$, is defined as follows:

Definition 3 *Let $\mathbf{A}, t, M$ be as above, and u any given node on the trdag t.*

- *If u is a leaf with $name(u) = a$, then $q \lhd_u q'$ iff whenever $a \to q \in \Delta$ we also have $a \to q' \in \Delta$;*
- *otherwise $q \lhd_u q'$ iff:*
  - *(i) $(name(u), M(Out_u)) \to q$ is an instance of a transition rule in Δ; i.e., Δ has a rule $(name(u), \omega) \to q$ such that $M(Out_u)$ is in ω;*
  - *(ii) there exists a map $\sigma_{q'} : Q \to Q$, such that:*
    - *- $\sigma_{q'}(q) = q'$, and the rule $(name(u), \sigma_{q'}(M(Out_u))) \to q'$ is also an instance of a transition rule in Δ;*
    - *- for any edge $\mathbf{e} : u \xrightarrow{i} u' \in Out_u$, we have: $M(\mathbf{e}) \lhd_{u'} \sigma_{q'}(M(\mathbf{e}))$.*

Definition 4 *Let $\mathbf{A} = (\Sigma, Q, F, \Delta)$ be any given TDA, and t any given* trdag. *A* run *of $\mathbf{A}$ on t is a* pair (r, M), *where $r : Nodes(t) \to Q$ and $M : Edges(t) \to Q$ are maps such that the following conditions hold, at any node u on t:*

(1) *if* $name(u) = f$*, then the rule* $(f, M(Out_u)) \to r(u)$ *is an instance of a transition rule in* Δ*;*

(2) *there is an incoming edge* $\mathbf{e} \in In_u$ *with* $M(\mathbf{e}) = r(u)$*; and for every* $\mathbf{e}' \in In_u$ *such that* $M(\mathbf{e}') = q' \neq q = r(u)$*, we have* $q \lhd_u^M q'$

A run (r, M) *is* accepting *on a* trdag t *iff* $r(\epsilon) \in F$*, i.e,* r *maps the root-node of* t *to an accepting state. A* trdag t *is accepted by a TDA iff there is an accepting run on* t*. The language of a TDA is the set of all* trdag*s that it accepts.*

Remark 1. i) Note that if t is a tree, then In_u is singleton at every non-root node u on t, so a run (r, M) of any TDA on t can be identified with its first component r; we get then the usual notion of runs of tree automata on trees.

Example 1. Over the unranked signature $\{a, f, g\}$ consider a TDA $\mathbf{A}$, with the following transitions:

$$\begin{array}{l} a \to p, \quad b \to q', \quad b \to p, \quad b \to q, \\ (a,\, p) \to q, \quad (a,\, q) \to p, \quad (a,\, q') \to q, \\ (g,\, q\,Q^*) \to q, \quad (g,\, p\,q) \to p, \\ (f,\, q\,p\,q) \to q_{fin}, \quad (f,\, p\,Q^*) \to q_{fin}, \end{array}$$

with $Q = \{p, q, q', q_{fin}\}$, and q_{fin} as the unique accepting state. An accepting bottom-up run of $\mathbf{A}$ on a tdag is depicted on the left of Figure 2, and on its right, the "same" run as seen on the tree equivalent of the tdag.

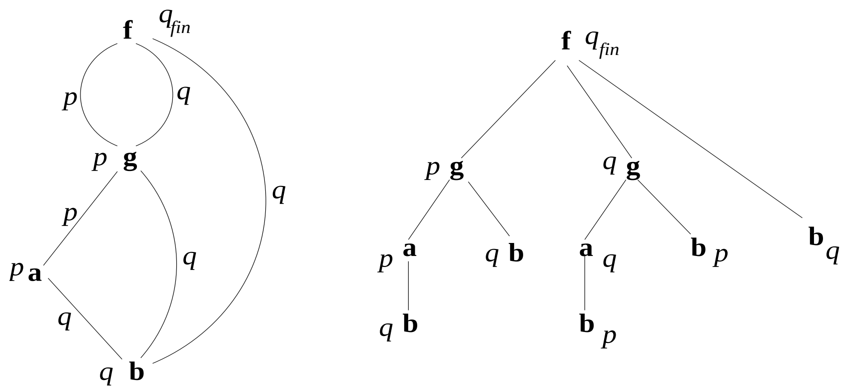


Fig. 2. A bottom-up accepting run of the TDA of Example 1 on a trdag, and the same seen on its tree equivalent.

A few comments on the above run may be of help: we start with assigning state q to the leaf node b, under r; the assignments of state q under M to all the incoming edges at this node b poses no problem; we can then assign state p to node a, and subsequently also p to the node g, under r, via the transition rule $(g, pq) \to p$; we then assign p under M to the first incoming edge at g; to assign state q under M to the second incoming edge at g, we just need to check that:

- for a map $\sigma : Q \to Q$ such that $\sigma(p) = q, \sigma(q) = p$, the rule $(g, \sigma(p)\sigma(q)) \to q$ is an instance of a transition rule of the TDA;

- for the outgoing edge $g \to a$, labeled with p by M, we have $p \lhd_a q = \sigma(p)$;

- for the outgoing edge $g \to b$, labeled with q by M, we do have $q \lhd_b p = \sigma(q)$;

reaching q_{fin} at the root-node is trivial via the last transition rule. (Note that we could have as well assigned p under M to the second incoming edge at g, with no conditions to check, then reach q_{fin}.)

Remark 1 (contd.). ii) Unlike the DAs of [5] or [1], the following bottom-up non-deterministic TDA: $a \to q_1,\ a \to q_2,\ f(q_1, q_2) \to q_a$, with q_0, q_1, q_a as states where q_a is accepting, has a non-empty language: as a TDA it accepts $f(a, a)$.

For a *deterministic* TDA, we have the following result (as expected):

Proposition 1 *Let* $\mathbf{A}$ *be a bottom-up deterministic TDA, and* t *any given trdag; then there is at most one run of* $\mathbf{A}$ *on* t.

Proof. Let Q be the set of states of $\mathbf{A}$, and $M : Edges(t) \to Q$ any given map assigning states to the edges on t. We shall show by induction that the hypothesis of determinism on $\mathbf{A}$ implies that, at any node u on t, the binary relation $\lhd_u^M = \lhd_u$ defined above (Definition 3), w.r.t. the map M, is the identity relation on the set Q. The proposition will then follow from conditions (1) and (2) on runs, cf. Definition 4; we will get, in particular, that for every incoming edge $\mathbf{e}$ at u, $M(\mathbf{e})$ must be the same as $r(u)$; so the run can be identified with its first component r (as on a tree).

The induction will be on a non-negative integer d_u, that we define at any node u of t – and refer to as its *height* on t– as the maximal number of arcs on t from u to the leaf nodes. If $d_u = 0$, then u is a leaf node; that $\lhd_u$ is the identity relation on Q in this case is immediate, from the determinism of $\mathbf{A}$, and the definition of $\lhd_u$. So, assume that $d_u > 0$, and let $v_1 \dots v_n$ be the string of all the children nodes of u on t. By the inductive hypothesis, for every $i, 1 \le i \le n$, the relation $\lhd_{v_i}$ is the identity relation on Q; it follows then, from the conditions (i) and (ii) on the relation $\lhd_u$ (Definition 3), that this latter must also be the identity relation on Q. ☐

We may now formulate the principal result of the first part of this paper:

Proposition 2 *i) A TDA accepts a* trdag t *if and only if it accepts the tree equivalent of* t.

ii) The emptiness problem for a TDA is decidable in time P *w.r.t. its number of states.*

iii) The uniform membership problem for a TDA is decidable in time NP *(resp. time* P*) w.r.t. its number of states, and the number of edges (resp. and the number of positions) on the given* trdag.

Proof. Let $\hat{t}$ be the tree equivalent of the trdag t, and $\mathbf{c}$ the natural surjective compression map from $Nodes(\hat{t})$ onto $Nodes(t)$.

Property i): For proving the 'only if' part, one uses the following reasoning, coupled with induction on the height function at the nodes of t (defined in the proof of the previous proposition): Let (r, M) be an accepting run of the given TDA on the trdag t; consider a node s on the tree equivalent $\hat{t}$, of which the node u on t is the image under the compression map $\mathbf{c}$; let $r(u) = q$ under the given run of the TDA on t; then, for every state q' of the TDA such that $q \lhd_u^M q'$, one can construct a partial run of the TDA –seen as a usual tree automaton– on the tree $\hat{t}$, climbing up from a leaf below s on $\hat{t}$ to the node s, and assigning the state q' to this node (for an illustrative example, see the tree to the right of Figure 2).

Proving the 'if' part of Property i) is a little more complex. We start with a given accepting run $\hat{\rho}$ of the given TDA, as a bottom-up tree automaton running in the usual sense on the tree $\hat{t}$; from this run $\hat{\rho}$, we shall construct a run (r, M) of the TDA on the trdag t, by an inductive, top-down traversal of the tdag t; for this top-down traversal, we will be using an integer valued function defined at any node u of t –and referred to as its *depth* on t– as the maximal number of arcs on t from the root node on t to the node u. We shall also use the fact that the nodes of $\hat{t}$ are in natural bijection with the set $Pos(t)$ of positions on t. The top-down construction of the run (r, M) is done by the following pseudo-algorithm, where $\mathbf{d}$ stands for the *maximal* depth on t at its leaf nodes.

```
BEGIN
/* define first r at the root node on t,
  and M on its outgoing edges */
r(ε_t) = ρ̂(ε_t̂);
For every outgoing edge e_j, 1 ≤ j ≤ k,
  at ε_t, set M(e_j) = ρ̂(ε.j);
i = 1; /* Now go down */
while (i < d) do {
  For every node u at depth i do {
     choose e ∈ In_u(t), and α ∈ pos_t(e)
     such that M(e) = ρ̂(α);
     set r(u) = M(e);
     For every e_j ∈ Out_u(t), 1 ≤ j ≤ m,
     outgoing from u, set M(e_j) = ρ̂(α.j);
  }
i = i + 1; }
END.
```

It is not difficult to check then, that by construction, the pair of maps (r, M) gives an accepting run of the TDA on the trdag t. (The reasoning is illustrated below.)

Properties ii) and iii) follow, in the ranked case, from the proof of i) and the results of TATA ([6]), Chapter 1; in the unranked case, one can either employ a reasoning based on reduction to the ranked case as in [10], or appeal directly to the results of [13]. (Note: the number of positions on a trdag is the same as the size of its tree equivalent.) $\square$

We illustrate here the reasoning employed in the proof of the 'if' part of assertion i) of the above proposition, with the tdag t of Example 1. We start with the run $\hat{\rho}$ on its tree-equivalent $\hat{t}$, as depicted to the right of Figure 2. At start, to the root node on t (at depth 0) is assigned the state q_{fin}, and to its three outgoing edges, are asigned the three states p, q, q respectively; at g, which is the only node on t at depth 1, we choose the first incoming edge (of position 1, and labeled with p by M), and set $r(u) = \hat{\rho}(1) = p$; the two outgoing edges at g on t have as positions the sets $\{11, 21\}, \{12, 22\}$ respectively; to these two outgoing edges at g on t, we assign the states that $\hat{\rho}$ assigns to the two sons of the node g at position 1 on $\hat{t}$, namely p, q respectively (this means in essence that we have 'selected' the positions 11 and 12 on the two outgoing edges at g on t); next, we go to depth 2 on t, where a is the unique node, to which we then have to assign the state $\hat{\rho}(11)$ that M has already assigned to its incoming edge; the rest of the reasoning is obvious, so left out.

Remark 2. Let $t \neq t'$ be two given trdags such that $Pos(t') = Pos(t)$, and there is a name preserving surjective map $\mathbf{c}'$ from $Nodes(t')$ onto $Nodes(t)$. We can then define t to be a compression, or compressed form, of t'; and refer to t' as an uncompressed equivalent of t, and to the surjective map $\mathbf{c}'$ on $Nodes(t')$ as a compression map. It is easily checked that t and t' have then the same tree-equivalent; and it folows from Proposition 2 above that any given TDA $\mathbf{A}$ accepts t if and only if it accepts t'. This means that it is legitimate to define the langauge of a TDA as the set of all tdags that it accepts (or trees that it accepts), or as the set of all trdags accepted, up to tree-equivalence.

3 Querying Compressed Documents: Preliminaries

Given a trdag t, one can naturally construct a regular tree grammar associated with t, which is *straightline* (cf. [4]), 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 $A_i \rightarrow f(A_{j_1}, \ldots, A_{j_k})$, 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 3 we get the following normalized grammar:

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

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.

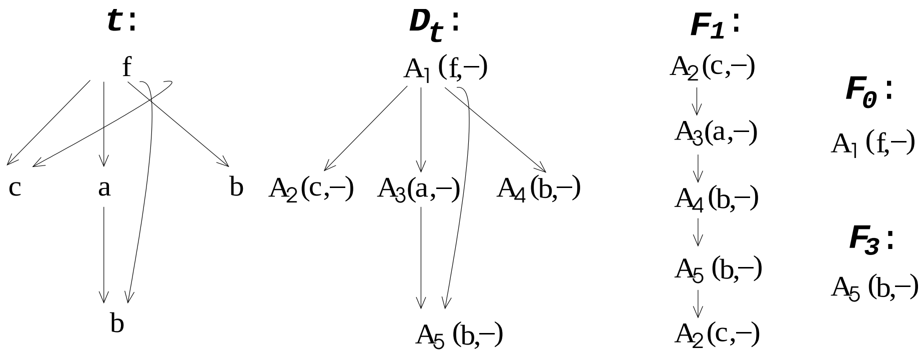


Fig. 3. trdag t, associated rlag $\mathcal{D}_t$, and chiblings of $\mathcal{L}_t$

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 $A_i \rightarrow f(A_{j_1}, \ldots, A_{j_k})$ 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 3.

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. [10, 19]): `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. [19]). 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 5 *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., [7]), 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 Appendix I. The following proposition results from Definition 5.

Proposition 3 *(1) For any set of nodes X on a trdag t, and any axis* A*, we have:* $\mathtt{A}(X) =$

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

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

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

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

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

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

Finally, following [2], 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 the 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 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 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 $\mathcal{D}_t$):
- $(\top', 1)$: $x = \sigma$, 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 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 either from the group $\{(\top, 0), (\top, 1), (\top', 1)\}$ or from the group $\{(\eta, 0), (\eta, 1), (s, 1)\}$; and this group is 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 $(\mathfrak{l}, x)$ 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 $(r(u), llab(v)) \to (\mathfrak{l}, x)$, 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, 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 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`]`

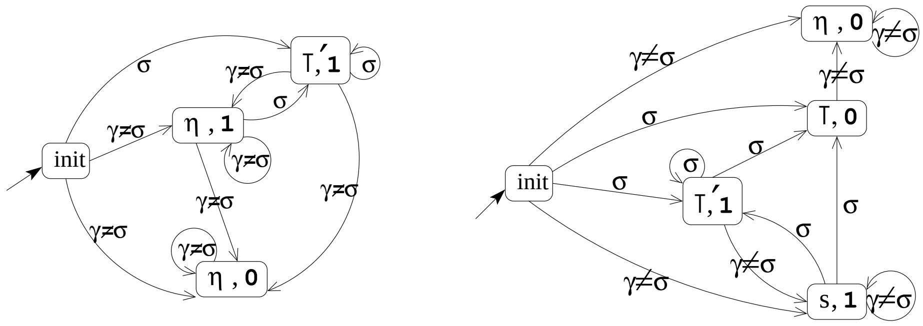


Figure 4 below illustrates the evaluation of $Q =$ `//*[following-sibling::`b`]`, on the trdag t of Figure 3. We first use the automaton for the basic query `//*[following-sibling::`σ`]` with $\sigma = b$, and then the automaton for `//*[self::`σ`]` with $\sigma = s$. The sub-trdag of t, formed of nodes corresponding to those of $\widehat{r}(\mathcal{D}_t)$ with labels having boolean component 1, contains all the answers to Q on t.

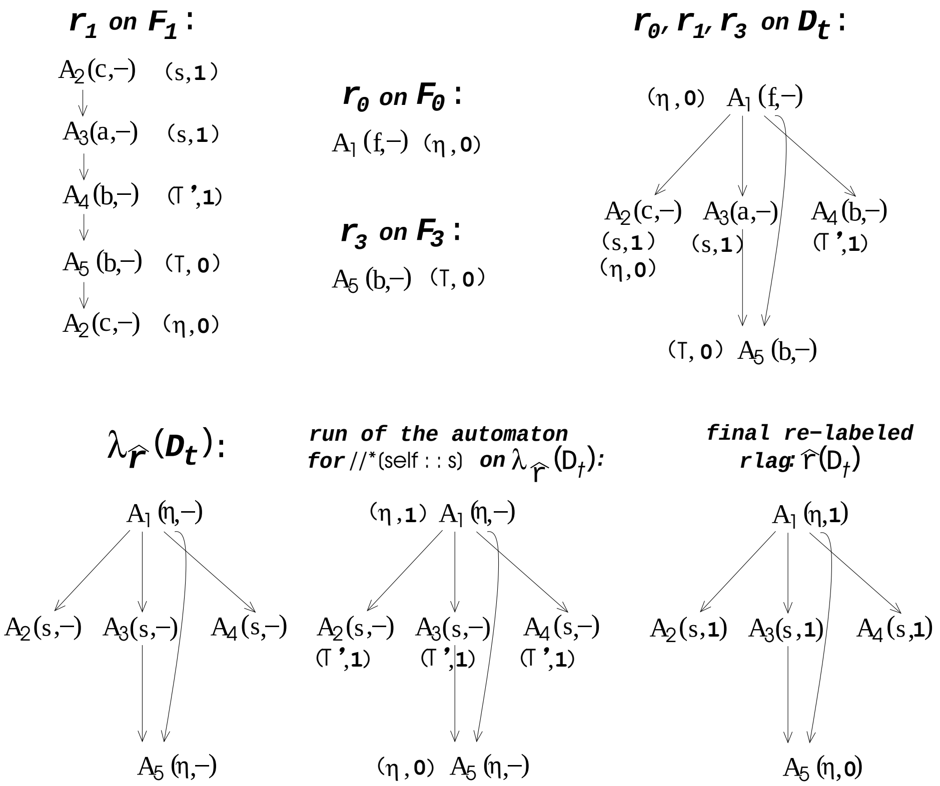
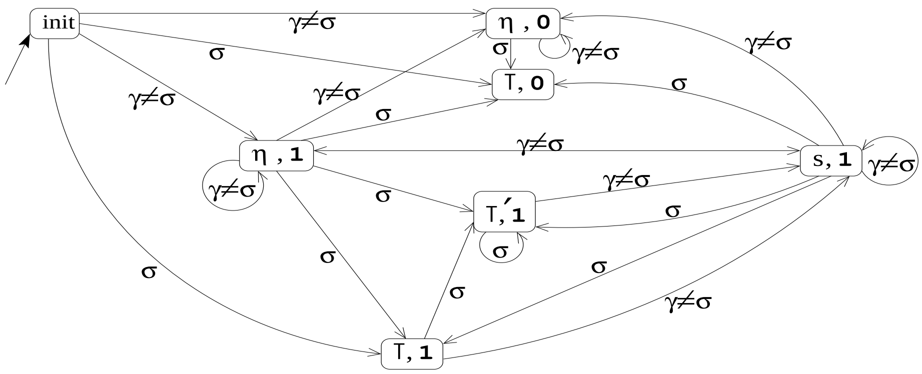
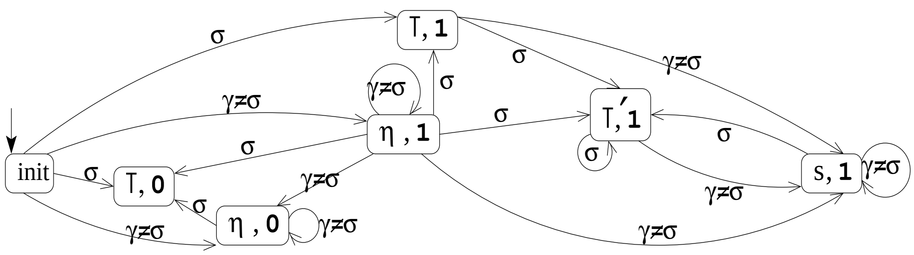


Fig. 4.

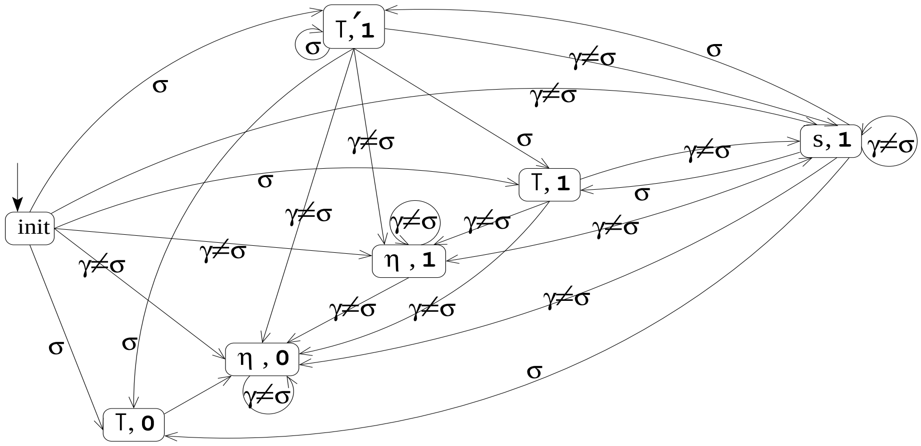
- <u>Automaton for the query //*[parent::σ]</u>



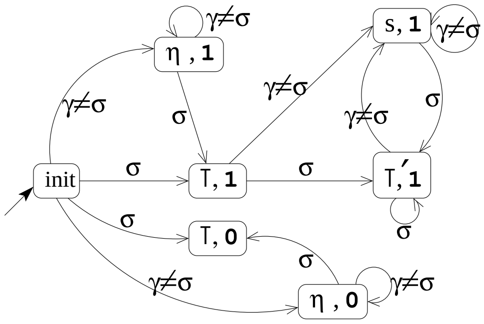
- <u>Automaton for the query //*[ancestor::σ]</u>



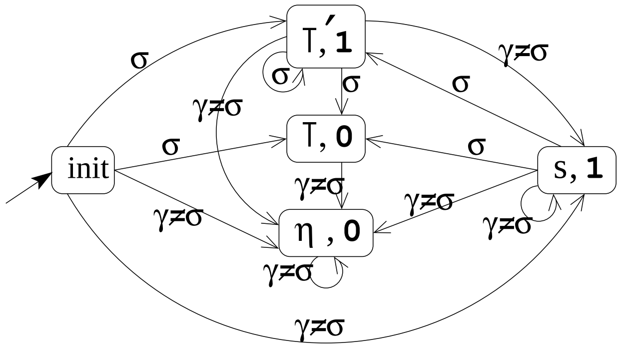
• Automaton for the query //*[child::σ]



• 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 $(init, llab(u)) \to$ 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$(w), llab(v)) \to$ 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. $\square$

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* $\mathbb{L}$ *on* $Nodes(\mathcal{G})$*, defined as* $\mathbb{L}(u) = r(u)$ *for any node* u*, is correct with respect to the semantics of subsection* 4.1*.*

Proof. Let us suppose that the labeling $\mathbb{L}$ 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 is not basic. We propose to evaluate such a query incrementally. For this, it suffices to consider queries that are of the form `//*[A::`x *`conn`* `A'::`x'`]`, where $conn \in \{and, or\}$, or of the form `//*[`$\mathtt{A}_1$`::*[`$\mathtt{A}_2$`::`σ`]]`. For those of the former type, we observe first that the components in a disjunction (resp. conjunction) under a '*' can be evaluated separately. Indeed, 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. an *and*). 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 presented in the 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 5 below illustrates the above reasoning, for the evaluation of the composite query Q = `//*[self::`b *and* `parent::`a`]`, on the trdag t of Figure 3:

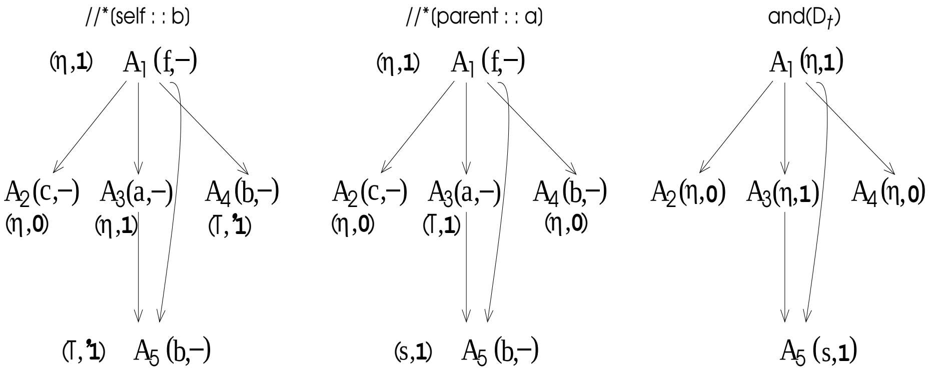


Fig. 5.

We next consider the queries of the form Q = `//*[`$\mathtt{A}_1$`::*[`$\mathtt{A}_2$`::`σ`]]`, with imbricated predicates. For their evaluation, 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 `//*[`$\mathtt{A}_2$`::`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 imbricated query Q on the dag t is then done by running the automaton for the basic outer query `//*[`$\mathtt{A}_1$`::`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 Deriving the Answer on the Tree-equivalent

Given a Core XPath query Q and its answer set on a trdag t, we show here how to derive the answer for the same query Q on the tree-equivalent $\hat{t}$ of t; this is of importance, since the "standard model" for an XML document (even when given in a compressed form) is generally considered as the tree representation of the document.

We observe, to start with, that the answer set for Q on t is in general a superset of the answer set for Q on the tree-equivalent $\hat{t}$. This can be so for the following two reasons:

(i) If a certain node u on t is selected by Q, not all of the nodes u' on $\hat{t}$, that are 'lifts' of u under the compression map $\mathbf{c}$ on $Nodes(\hat{t})$, may answer the query Q on the tree $\hat{t}$, even when Q is a basic query. For instance, consider the basic query `//*[parent::`a`]`; on the fully compressed tdag $f(a(c), b(c))$, the (unique) node named c is an answer; it has two c-named nodes as lifts on the tree-equivalent $\hat{t}$, of which only one is an answer for the query.

(ii) A node u on a trdag t may answer a composite query Q, but none among the lifts of u on $\hat{t}$ may answer the same query Q on the tree $\hat{t}$. For instance, the unique c-named node on the compressed tdag $f(a(c), b(c))$ answers the query `//*[parent::`a ***and*** `parent::`b`]`, but there is no node on the tree-equivalent answering this query.

Actually, such situations arise only for queries involving the upward axes `parent, ancestor`, which define relations that are less trivial on trdags than on trees. We can formulate this observation more precisely, as follows:

Lemma 1. *Let* `A` *be one of the axes* `self`, `child`, `descendent`*, Q the basic query* `//*[A::`x`]`*, t any given trdag, $\hat{t}$ its tree-equivalent, u any given node on t, and $u' \in \mathbf{c}^{-1}(u)$ any node lift of u on $\hat{t}$. Then:*

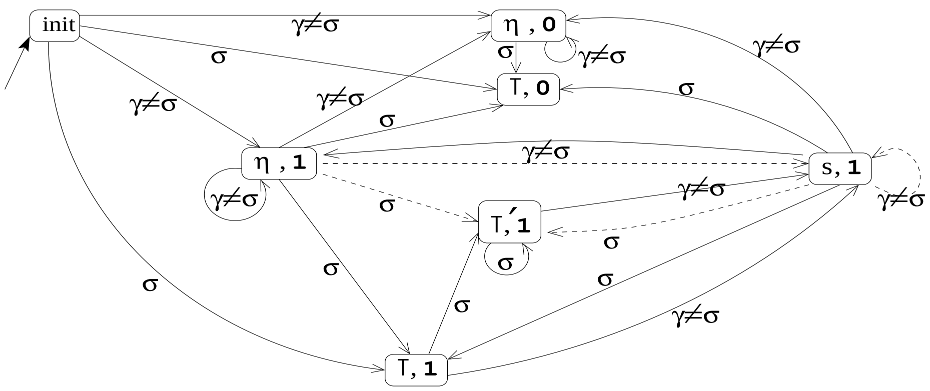
• *wrt the maximal priority runs of the automaton for the axis* `A`*, respectively on $\mathcal{D}_t$ and $\mathcal{D}_{\hat{t}}$, the nodes u on t, and u' on $\hat{t}$, get labeled by the same ll-pair;*

• *in particular, the node u answers Q on t* if and only if *the node u' answers the same query Q on the tree $\hat{t}$.*

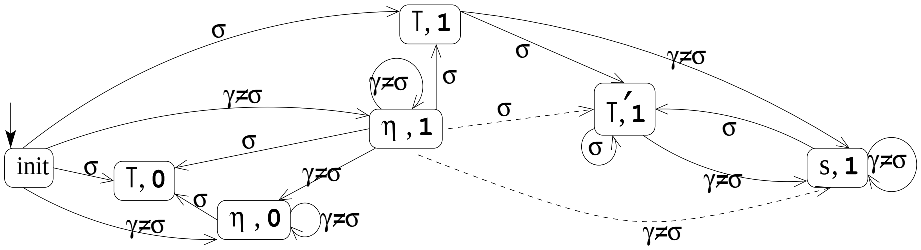
Proof. Follows by observing that the semantics of Section 4.1 have been defined in a manner which is top-down, and that the compression map $\mathbf{c} : Nodes(\hat{t}) \to Nodes(t)$ maps the set $Nodes(\hat{t}|_{u'})$, of nodes below u' on $\hat{t}$, onto the set of nodes of the sub-trdag $t|_u$. □

The above lemma is a first step towards the objective of this section. As a second step, we propose to distinguish, on the automata constructed above for the two queries `//*[parent::`σ`]`, `//*[ancestor::`σ`]`, the transitions that will never be fired on a tree; such as, e.g., the one from state $(\eta, 1)$ to state $(\top', 1)$. (Note: for this transition to be firable, we have to reach a node corresponding to a σ-named node on the trdag, which must then also have as (unique) parent on the tree –i.e., the node from which the transition is to be fired– a σ-named node; this parent node cannot correspond then to a node labeled with $(\eta, 1)$.) Such transitions that are not firable on a tree) will be depicted with *dotted arrows* on the automaton; the transitions with full arrows are then the ones that are firable both on trdags and on trees. The two automata thus revised are as follows:

• <u>Automaton for the query `//*[parent::`σ`]` -revised</u>



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



The next step towards our objective of this section consists in completing a maximal priority run r of the automaton for any given basic query, by associating to a current node u on $\mathcal{D}_t$, a subset of $Pos_t(u)$ (remember: $Nodes(t)$ and $Nodes(\mathcal{D}_t)$ are in natural bijection), denoted as $P_r(u)$, and defined as follows:

• case u selected (i.e. the label of u under r is $(s,1)$ or $(\top',1)$: we set $P_r(u) = Pos_t(u) \smallsetminus \cup_{\alpha,i}\{\alpha.i \mid \alpha.i \in Pos_t(u)\}$, the union being taken over the positions α of parent nodes v on t such that the transition from v to u is *dotted*;

• case u not selected (i.e. the label of u under r is neither $(s,1)$ nor $(\top',1)$: we set here $P_r(u) = Pos_t(u)$.

A run r completed in this manner will be denoted in boldface type as $\mathbf{r}$, giving thus a map $\mathbf{r} : Nodes(t) \to L \times \{0,1\} \times Pos(t)$, defined by $u \mapsto (r(u), P_r(u))$.

In order to derive the answer to a composite query Q on the tree-equivalent of t, from the answer for Q on t, we need naturally to complete the functions AND and OR of Section 6, by adding component giving the selected positions for a query Q which is a conjunction or disjunction of two sub-queries Q_1, Q_2. These completed functions, again denoted boldface as **AND** and **OR**, or defined below in a rather obvious manner (the indices $1,2$ correspond to the runs wrt the two queries, and l_1, l_2 stand for s or η; recall that η' stands for the llabel s):

$$\begin{aligned}
&\mathbf{AND}(u) = ((s,1),\ P_{r_1}(u) \cap P_{r_2}(u)) \ \text{ iff}\\
&\qquad\qquad \mathbf{r}_1(u) = ((l_1',1), P_{r_1}(u)) \ \text{ and } \ \mathbf{r}_2(u) = ((l_2',1), P_{r_2}(u));\\
&\mathbf{AND}(u) = ((\eta,0), Pos_t(u)) \ \text{ iff } \ r_1(u) = (l_1,0) \ \text{ and } \ r_2(u) = (l_2,0);\\
&\mathbf{AND}(u) = ((\eta,1), Pos_t(u)), \text{ otherwise.}
\end{aligned}$$

$$\begin{aligned}
&\mathbf{OR}(u) = ((s,1),\ P_{r_1}(u) \cup P_{r_2}(u)) \ \text{ iff}\\
&\qquad\qquad \mathbf{r}_1(u) = ((l_1',1), P_{r_1}(u)) \ \text{ and } \ \mathbf{r}_2(u) = ((l_2',1), P_{r_2}(u));\\
&\qquad = ((s,1),\ P_{r_1}(u)) \ \text{ iff } \ r_1(u) = (l_1',1) \ \text{ and } \ r_2(u) \neq (l_2',1);\\
&\qquad = ((s,1),\ P_{r_2}(u)) \ \text{ iff } \ r_2(u) = (l_2',1) \ \text{ and } \ r_1(u) \neq (l_1',1);\\
&\mathbf{OR}(u) = ((\eta,0), Pos_t(u)) \ \text{ iff } \ r_1(u) = (l_1,0) \ \text{ and } \ r_2(u) = (l_2,0);\\
&\mathbf{OR}(u) = ((\eta,1), Pos_t(u)), \text{ otherwise.}
\end{aligned}$$

Example 2. We evaluate the query $Q =$`//*[ancestor::`b` [parent::`c`]]` on the trdag t presented to the left of Figure 6; the standard form of this query is $Q =$`//*[ancestor::*[self::`b` and parent::`c`]]`; and its answer consists of all the nodes having an ancestor b with parent c. But we want here to obtain the 'same' answer for Q on t and on its tree-equivalent $\hat{t}$ presented to the right of Figure 6. To find such an answer, it is necessary to use the revised automata for the `parent` and `ancestor` axes. For ease of comprehension, we illustrate the evaluation of Q directly on the trdag t (and not on the rlag $\mathcal{D}_t$) –it is possible because, for this document the trdag t and its rlag $\mathcal{D}_t$ are isomorphic. Note that each node u of t is represented by its name and the set of positions of the nodes on $\hat{t}$ that are lifts of u.

First, look at Figure 7 where we have presented the evaluation of Q using the non-revised automata. We obtain then an answer on t selecting nodes a and g (which are the nodes of t having an ancestor b with parent c); but, if we unfold this answer, we obtain the tree with as selected nodes: a at positions $11, 211$, and

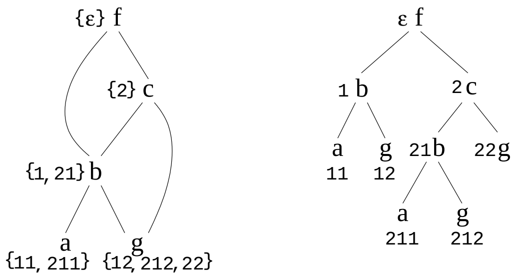


Fig. 6. tdag t and its tree-equivalent $\hat{t}$

g at positions $12, 212, 22$; obviously this set of nodes is *not* the answer for Q on $\hat{t}$: indeed, only the nodes a at position 211 and g at position 212 have ancestor b (position 21) with parent c (position 2) on the tree $\hat{t}$.

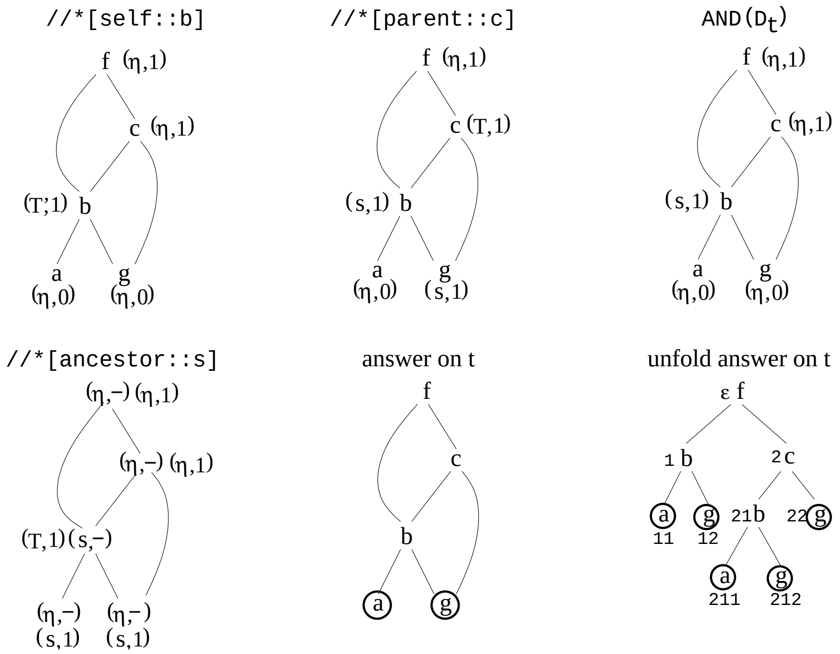


Fig. 7. Evaluation of Q on t using the non-revised automata

The Figure 8 presents the evaluation of Q on t using the revised automata. As before, first we evaluate in parallel the inner queries `//*[self::`b`]` and `//*[parent::`c`]`, but this time, to any node on t, the runs of the revised automata associate not only a state, but also a *set of (edge-)positions*, comprising exactly those corresponding to the productions of the revised automata that are represented by full edges. We find next the answer for conjunction using the revised function **AND**. This way we have obtained a relabeled rlag with the names of nodes in $\{(\eta, -), (s, -)\}$, and on this rlag we evaluate the outer query `//*[ancestor::s]`. Using the revised automata, we thus obtain the answer for Q on t restricted to a set of positions; by unfolding this answer, we get exactly the answer for Q on the tree-equivalent $\hat{t}$ of t.

Remark 3. The algorithm given in [7], for the evaluation of (Core) XPath queries on a compressed dag t, actually takes a *given position* on t as a parameter; this explains that their method works indifferently on the unfolded tree $\hat{t}$, or on

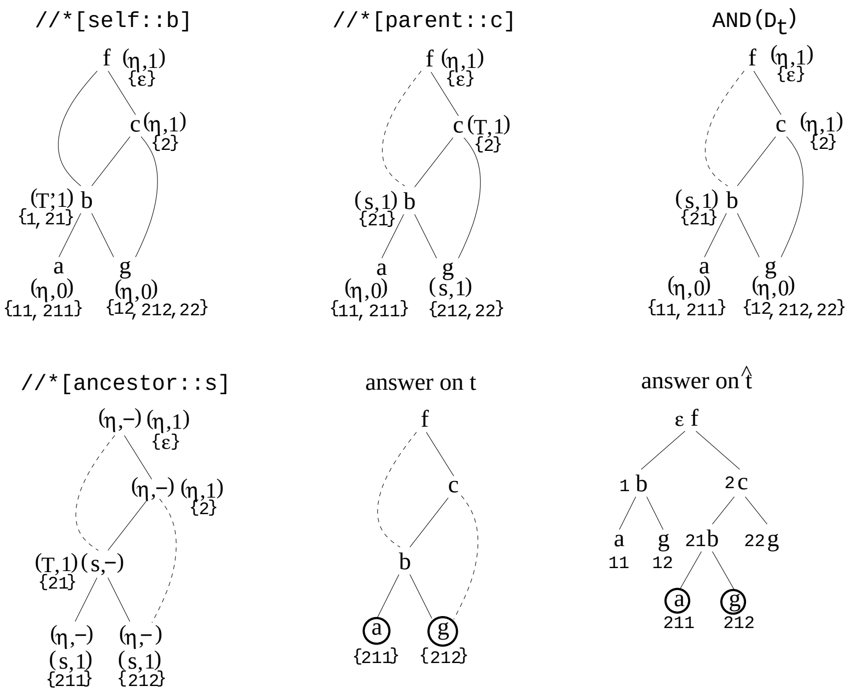


Fig. 8. Evaluation of Q on t using the revised automata

the dag t (cf. Algorithm 4.1, and Theorem 10, [7]). The method we have presented is more general, in that it returns *all* the positions on t (or $\hat{t}$) answering the query.

8 Conclusion

Our concern in this paper has been two-fold. The first part addressed the problem of running any bottom-up (unranked) tree automaton indifferently on a tree or on any of the dags obtained from the tree by full or partial compression; this gave rise to the notions of Tree/Dags (trdags) and of Tree/Dag automata. The second part of the paper addressed the issue of retrieving information from a trdag representing an XML document possibly given in a compressed form. (Note: Information retrieval from compressed structures, without having to uncompress them, is a field of active research; cf. e.g., [17, 11].) Limiting our concern here to the evaluation of queries formulated in terms of XPath axes, and more precisely 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 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 Appendix II; 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 $\mathcal{O}(n^2)$ if t is a tree, the relation *Parents* becoming trivial. (Note: our method of evaluation a priori gives the answers for the given query Q on the given trdag t, but we have shown how one can derive the answer set for Q on the tree-equivalent of t.)

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. S. Anantharaman, P. Narendran, M. Rusinowitch, *Closure Properties and Decision Problems of Dag Automata*, In Information Processing Letters, 94(5):231–240, 2005.
2. 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.
3. G. Busatto, M. Lohrey, S. Maneth, *Grammar-Based Tree Compression.* EPFL Technical Report IC/2004/80, `http://icwww.epfl.ch/publications`.
4. G. Busatto, M. Lohrey, S. Maneth, *Efficient Memory Representation of XML Documents.* In Proc. DBPL'05 (to appear), LNCS 3774, Springer-Verlag, 2005.
5. W. Charatonik, *Automata on DAG Representations of Finite Trees*, Technical Report MPI-I-99-2-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany.
6. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, *Tree Automata Techniques and Applications*, `http://www.grappa.univ-lille3.fr/tata/`
7. M. Frick, M. Grohe, C. Koch, *Query Evaluation of Compressed Trees*, In Proc. of LICS'03, IEEE, pp. 188–197.
8. G. Gottlob, C. Koch, *Monadic Queries over Tree-Structured Data*, In Proc. of LICS'02, IEEE,
9. 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.
10. 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.
11. M. Lohrey, *Word problems and membership problems on compressed words* In SIAM Journal of Computing, 35(5):1210-1240, 2006.
12. M. Marx, *XPath and Modal Logics for Finite DAGs.* In Proc. of TABLEAUX'03, pp. 150–164, LNAI 2796, 2003.
13. W. Martens, F. Neven, *On the complexity of typechecking top-down XML transformations*, In Theoretical Computer Sc., 336(1): 153–180, 2005.
14. M. Murata, A. Tozawa, M. Kudo, *XML Access Control Using Static Analysis*, In Proc. of the 10th ACM Conf. on Computer and Communications Security (CCS'03), pp.73–84, ACM, 2003.
15. F. Neven, *Automata Theory for XML Researchers*, In SIGMOD Record 31(3), September 2002.
16. F. Neven, T. Schwentick, *Query automata over finite trees*, In Theoretical Computer Science, 275(1–2):633–674, 2002.
17. W. Rytter, *Compressed and fully compressed pattern matching in one and two dimensions*, In Proceedings of the IEEE, 88(11):1769–1778, 2000.
18. J.W. Thatcher, J.B. Wright, *Generalized finite automata theory with an application to a decision problem of second-order logic*, In Math. Syst. Theory, 2(1):57–81, 1968.
19. World Wide Web Consortium, *XML Path Language (XPath Recommendation)*, `http://www.w3c.org/TR/xpath/`

Appendix I: From Canonical Forms to Standard Forms

We stick to the notations of Section 3.1. Given any canonical XPath 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:

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

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

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

This translation procedure is of complexity linear with respect to the total number of location steps (i.e. of the form $\texttt{axis::}\sigma$) that appear in Q.

Appendix II: Constructing the Maximal Priority Run

Given a trdag t, and a basic non-sibling query Q we give here the algorithm constructing the maximal priority run of the automaton $\mathbf{A}_Q$ on $\mathcal{D}_t$. (It is trivial for the sibling queries.) Let n be the number of nodes on $\mathcal{D}_t$; The idea is to construct a directed acyclic graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, representing all possible (complete) runs of $\mathbf{A}_Q$ on $\mathcal{D}_t$, and at the same time choose the one of maximal priority. The set $\mathbb{V}$ of vertices of $\mathbb{G}$ are elements of the form $A_{i,(l,x)}, 1 \le i \le n$, where A_i is a non-terminal of $\mathcal{L}_t$, and (l, x) is an ll-pair such that (l, x) is in $States(\mathbf{A}_Q)$ such that there exists a (complete) maximal priority run r of $\mathbf{A}_Q$ on $\mathcal{D}_t$ with $r(A_i) = (l, x)$. The algorithm has four steps as described below (where $llab(A_i)$ is the first component of the current label at the node on $\mathcal{D}_t$ named A_i).

Step i): For every non-terminal A_i of $\mathcal{L}_t$, $1 \le i \le n$ compute the set $Parents(A_i)$ of its parent non-terminals. To render the presentation uniform, we shall introduce a 'fictive' symbol A_0, and set $Parents(A_1) = \{A_0\}$.

The cost of this step is $\mathcal{O}(n^3)$: For every A_i we have to check if A_i is in $Sons(A_j)$ for $j \in \{1, \ldots, i-1\}$. The maximal size of $Sons(A_j)$ can be $(n-j)$, thus, for A_i we have $\sum_{j=1}^{i-1}(n-j) = (i-1)n - \sum_{j=1}^{i-1} j \in \mathcal{O}(i(n-i))$. In total we get: $\sum_{i=1}^{n} i(n-i) \le n \sum_{i=1}^{n} i \in \mathcal{O}(n^3)$.

Stpe ii): Construct the set of vertices and the arcs of the graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$.

For every $i \in \{1, \ldots, n\}$ we have at most three different (l, x) determined by $llab(A_i)$, so constructing the vertices costs $\mathcal{O}(3n) = \mathcal{O}(n)$.

Next, a pair $(A_{j,(l',x')}, A_{i,(l,x)})$ will be an arc in $\mathbb{E}$ iff $A_{j,(l',x')}, A_{i,(l,x)}$ are in $\mathbb{V}$, and the rule $((l', x'), llab(A_i)) \to (l, x)$ is a transition of $\mathbf{A}_Q$. So, for every given vertex $A_{j,(l',x')} \in \mathbb{V}$, we construct the set $Target(A_{j,(l',x')})$ of indices of all non-terminals A_i, reachable (with some label), from $A_{j,(l',x)}$ on $\mathbb{G}$. The cost of this construction is in $\mathcal{O}(n^2)$: indeed, for every vertex $A_{j,(l',x')}$ and for every $A_{i,(l,x)} \in \mathbb{V}$ such that $A_j \in Parents(A_i)$, we have to check if the former is reachable from the latter; that gives us $3\#(Parents(A_i))$ transitions of $\mathbf{A}_Q$ to check, so, on the whole, we get: $3\sum_{i=1}^{n} 3\#(Parents(A_i)) = 9\sum_{i=1}^{n}(i-1) \in \mathcal{O}(n^2)$.

Step iii): Eliminate the incomplete runs.

Consider any two given $i, j \in \{1, \ldots, n\}$, and suppose $A_{j,(l',x')}$ is a vertex such that $j \in Parents(A_i)$ but $i \notin Target(A_{j,(l'',x'')})$; this means A_i is not reachable from the former vertex if we associate the label (l', x') to A_j; and this implies that the vertex $A_{j,(l',x')}$ is incorrect; we remove from $\mathbb{G}$ all incorrect vertices, and all the incoming arcs at such vertices (by a bottom-up traversal of $\mathbb{G}$).

Finding all the incoming arcs at a given $A_{i,(l,x)}$, costs $3\#Parents(A_i) = 3(i-1)$ checks; so, the total cost of this step amounts to $3\sum_{i=1}^{n} 3(i-1) \in \mathcal{O}(n^2)$.

Step iv): Choose the maximal priority run.

Begin by setting $r(A_0) = init$; then, for every A_i, $i \in \{1, \ldots, n\}$, associate a pair (l, x) such that: $A_{i,(l,x)} \in \mathbb{V}$ and (l, x) is of maximal priority between all possible pairs (l', x') such that $A_{i,(l',x')} \in \mathbb{V}$. The cost of this last step is in $\sum_{i=1}^{n} 3(i-1) \in \mathcal{O}(n^2)$.

The total cost of the algorithm is therefore $\mathcal{O}(n^3)$.

Given:

The dependency dag $\mathcal{D}_t$, n = its number of nodes;
$A_i, 1 \leq i \leq n$: the non-terminals of $\mathcal{L}_t$ naming the nodes of $\mathcal{D}_t$;
$\mathbf{A}_Q$ the automaton for the basic query Q,
$States$ = the set of states of $\mathbf{A}_Q$, Δ = the set of transition rules of $\mathbf{A}_Q$.

BEGIN :
/* Step i): Construct the relation $Parents$ */
$Parents(A_1) := \{A_0\}$; for all $i \in \{2, \ldots, n\}$ $Parents(A_i) := \emptyset$;
For $i \in \{2, \ldots, n\}$ **do**
For $j \in \{1, \ldots, i-1\}$ **do**
If $A_i \in Sons(A_j)$ $Parents(A_i) := Parents(A_i) \cup \{A_j\}$;
od; **od**;

/* Step ii): Construct the vertices $\mathbb{V}$, the arcs $\mathbb{E}$, and the $Target$ sets */
$\mathbb{V} := \{A_{0,init}\} \cup \{A_{i,(l,x)} \mid 1 \leq i \leq n,\ l$ defined by $llab(A_i)$, $(l, x) \in States\}$
$\mathbb{E} := \emptyset$; **For all** $A_{i,(l,x)} \in \mathbb{V}$ $Target(A_{i,(l,x)}) := \emptyset$;
For all $A_{i,(l,x)} \in \mathbb{V}$, with $i \geq 1$
For all $A_j \in Parents(A_i)$
If $A_{j,(l',x')} \in \mathbb{V}$ and $((l', x'), llab(A_i)) \to (l, x) \in \Delta$ {
$\mathbb{E} := \mathbb{E} \cup \{(A_{j,(l',x')}, A_{i,(l,x)})\}$;
$Target(A_{j,(l',x')}) := Target(A_{j,(l',x')}) \cup \{i\}$;
}

/* Step iii): Eliminate the incomplete runs, via a bottom-up search */
$i := n$;
(#) **For all** $A_{i,(l,x)}, A_{i,(l',x')} \in \mathbb{V}$
If $Target(A_{i,(l,x)}) \subsetneq Target(A_{i,(l',x')})$ {
$\mathbb{V} := \mathbb{V} \setminus \{A_{i,(l,x)}\}$;
For all $A_{k,(l',x')} \in \mathbb{V}$ such that $(A_{k,(l',x')}, A_{i,(l,x)}) \in \mathbb{E}$ {
$\mathbb{E} := \mathbb{E} \setminus \{(A_{k,(l',x')}, A_{i,(l,x)})\}$;
If $\mathbb{E} \cap \{(A_{k,(l',x')}, A_{i,(l,y)}) \mid (l, y)$ defined by $llab(A_i)\} = \emptyset$
$Target(A_{k,(l',x')}) := Target(A_{k,(l',x')}) \setminus \{i\}$;
}
}
$i := i - 1$;
If $i \geq 1$ GOTO (#);

/* Step iv): Construct now the maximal priority run, top-down */
$r(A_0) := init$; $i := 1$;
(**$**) $r(A_i) := max\{(l, x) \mid A_{i,(l,x)} \in \mathbb{V}\}$;
If $i < n$, $\{i := i + 1$; GOTO (**$**)$\}$;
Else RETURN $\{r(A_1), \ldots, r(A_n)\}$;
END.

Appendix III: A Complete Example

1) We evaluate the query $Q =$ `/descendant::*[descendant::`b `[parent::`a`]]` on the partially compressed document t, given to the left of Figure 6. Note that we want to select every node having some descendant b with parent a. To start with, we first translate Q into standard form, as:

$Q =$ `//*[descendant::*[self::`b *and* `parent::`a`]]`

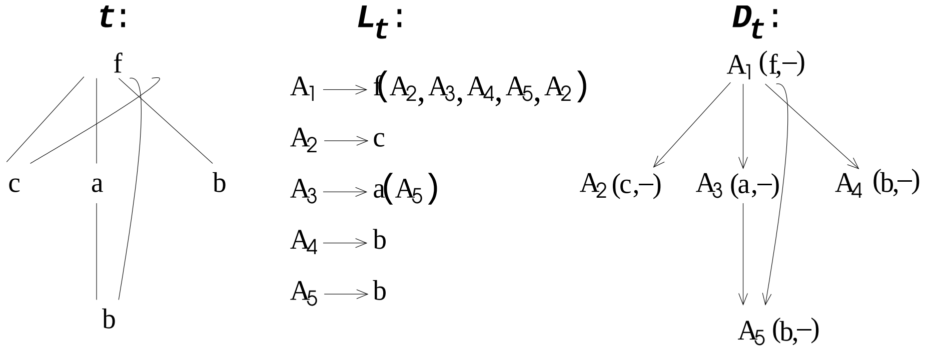


Fig. 9. Document t, its normalized Grammar $\mathcal{L}_t$, and the Dependency rlag $\mathcal{D}_t$

Figure 7 represents, to the left, the rlag $\mathcal{D}_t$ labeled by the automaton for the query `//*[self::`b`]`; to the middle, the same rlag labeled by the automaton for the query `//*[parent::`a`]`; and to the right, the rlag $\mathcal{D}'_t$ re-labeled for the conjunction, as explained in Section 6.

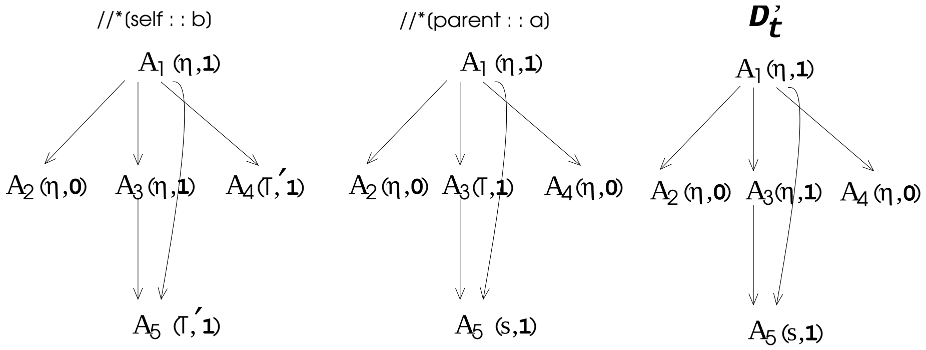


Fig. 10.

Figure 8 shows, to the left, the rlag obtained by re-labeling $\mathcal{D}'_t$ with the run of the automaton for the query `//*[descendant::`s`]`; and to the right, the (minimal) sub-rlag of $\mathcal{D}_t$ formed of nodes marked now by ll-pairs with boolean component 1, and the corresponding answer for the query Q on the document t. This final sub-rlag of $\mathcal{D}_t$ is obtained by cutting out the nodes where the ll-pairs attached have boolean component 0.

2) On the same document t as above, we consider now the following standard form query: $Q' =$ `//*[child::`b *or* `following-sibling::`b`]`

To the left of Figure 9 is the rlag $\mathcal{D}_t$ labeled by the run of the automaton for `//*[child::`b`]`; and to the right, the labeling of the 3 chiblings of $\mathcal{D}_t$, by the run of the automaton for `//*[following-sibling::`b`]`.

The two rlags $\mathcal{D}'_t, \mathcal{D}''_t$ of Figure 10 below, are then obtained by applying the re-labeling functions respectively lab_r and $\lambda_{\hat{r}}$ (of subsection 4.2) for these

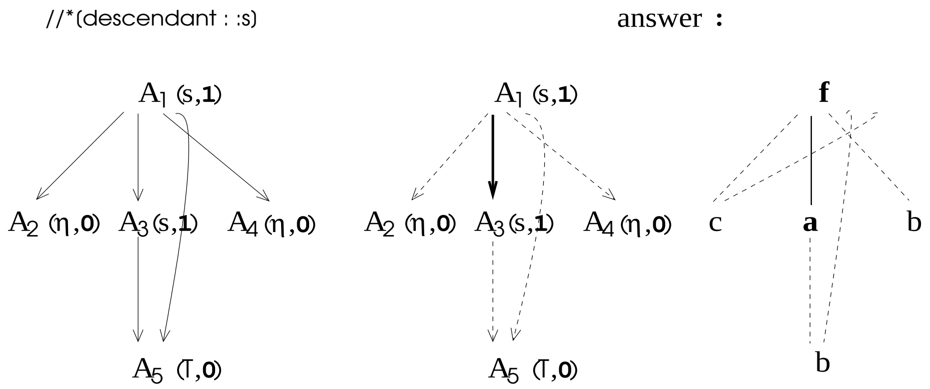


Fig. 11.

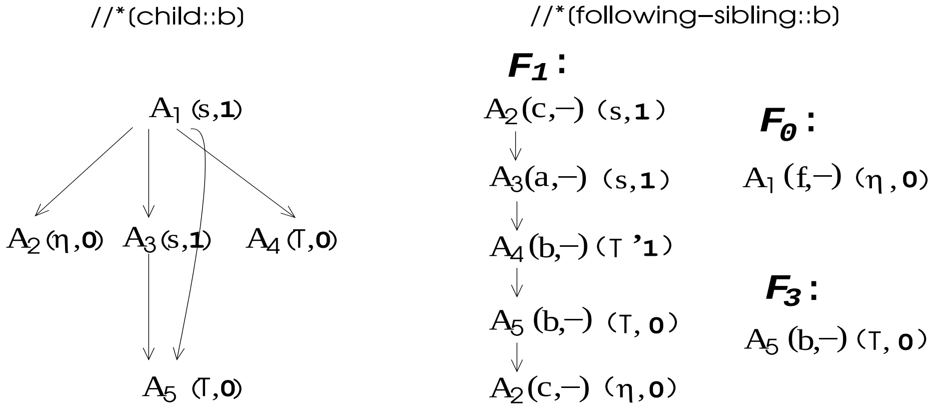


Fig. 12.

respective runs. The rlag $\mathcal{D}_t'''$ to the right is obtained by the run of automaton for `//*` `[self::`s`]` on $\mathcal{D}_t''$.

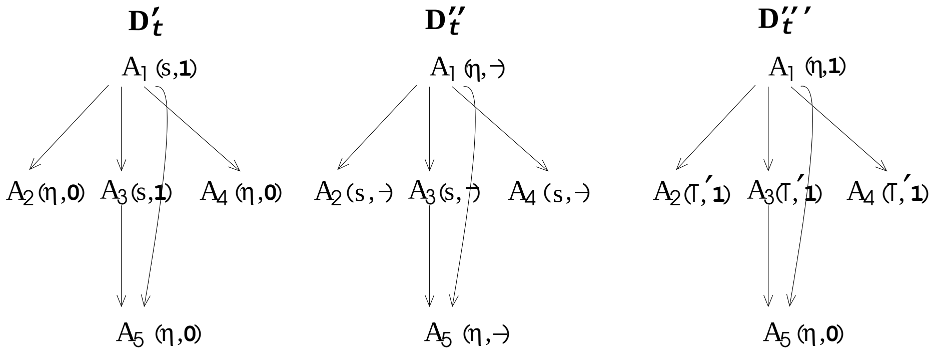


Fig. 13.

The Figure 11 presents the final answer for our query obtained by applying the function OR from section 6.

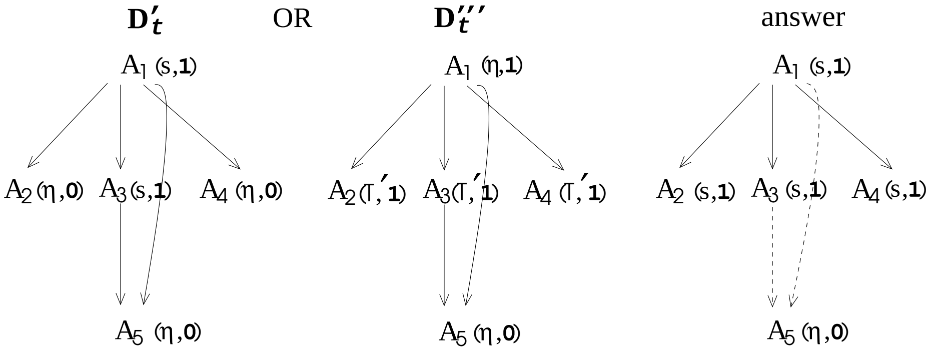


Fig. 14.