4 rue Léonard de Vinci
BP 6759
F-45067 Orléans Cedex 2
FRANCE
http://www.univ-orleans.fr/lifo

# Rapport de Recherche

## A Clausal View for Access Control and XPath Query Evaluation

B. FILA, LIFO (Orléans)
S. ANANTHARAMAN, LIFO (Orléans)

Rapport N$^o$ **2007-12**

# A Clausal View for Access Control and XPath Query Evaluation

Barbara Fila, Siva Anantharaman

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

**Abstract.** We show that access control policies on semi-structured, XML-like, documents can be handled naturally by a clausal view, for positive queries in the XPath format. The evaluation of any such query $Q$, on any given document $t$, is done step-wise, under unambiguous runs of a transition graph $G_Q$, that we associate with any XPath query $Q$ given in canonical form; the runs of the graph $G_Q$ on $t$ are defined in terms of clauses, controlled by appropriate set(s) of constraints.

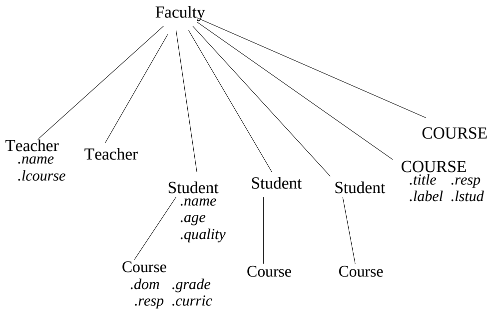
*Keywords*: XML, XPath, query, access control, constraints, clauses.

## 1 Introduction

We begin by illustrating, on a concrete example, our view for modeling access control on XML documents in terms of clauses (some familiarity with XML is assumed). The data stored at the nodes of an XML document contains in general a great variety of information, all of which is not meant to be accessed by all who consult the document. Various methods have been envisaged for controlling the access to data (cf. e.g., [5]). We mention two of them here. The approach of [10] is based on attributing access keys to nodes, or more generally to subsets of attributes at the nodes; the keys could depend on a notion of category to which the consultant belongs. [3] proposes a different, though not unrelated, approach: to every category of consultants is associated a set of first-order clauses, expressing constraints, such as the following: if the user consulting the document is of a certain category, then (s)he is not allowed to see both the data stored by $att_1$ and by $att_2$. We may also mention the view proposed in [11], where some special labels are attached to the nodes of the document, such as $-r, -R$, to signify 'local' (resp. 'recursive') access denial.

Our proposal in this paper is to model access control policies as first-order clauses. (It is shown in Section 5.1 how to model in terms of clauses, the key attribution approach mentioned above, as well as the approach of [11]). The clauses modeling access control will be subject, in general, to some constraints referred to as *scope constraints*. A situation such as the one described in the following example, illustrates well the ideas that we want to develop.

**Example 1**. Suppose given an (XML-format) database "Faculty", with as children nodes "Teacher", "Student", and "COURSE". A teacher has as attributes his/her name, and a list of courses (s)he is in charge of. A student has children nodes named "Course", each course having as attributes its domain, the name of the teacher giving the course, the grade obtained, and the curriculae for which the student follows the course; the student node also has as attributes his/her name, age and quality (full- or part- time student). Every COURSE node has as attributes the title of the course, its administrative label, the name of the teacher in charge, and the list of students following the course.




One could envisage three categories of users consulting this document: *Adm* (resp. *Acad*) for administrative (resp. academic) staff, and *Stud* for students. Here is an access control policy on this document, formulated in plain text:

*Adm*: a user of this category may not have access both to the name of a student and to the grade obtained in any of the course the student follows; but access to either one of these two data is allowed.

*Acad*: a user of this category is allowed access to the name of a student, and to the grade obtained by the student in any course, *iff* the user is the person giving the course.

*Stud*: a user of this category is not allowed to get access to the data stored on any student other than himself/herself.

To model these policies, we first agree to consider any category as a unary predicate, evaluating to 'true' only on the identifiers assigned to each user of the category, that we assume to be the name of the user concerned. The policy for *Adm* is then modeled as the following pure negative clause:

(1): $\leftarrow Adm(*),\ \text{Student.name},\ \text{Course.grade},$

$$[\text{Student.name } \texttt{child} \text{ Course.grade}]$$

where '$*$' stands for any identifier for a user of the category *Adm*. The intended meaning is that, for a user of category *Adm*, the access policy is violated if the knowledge (s)he gains by consulting the database (possibly repeatedly), contains the name of a student, as well the grade obtained by the student in a course; the scope constraint, inside the square brackets, says that the attributes 'name' and 'grade' are at two `child`-related nodes with tagnames 'student' and 'course', respectively. Note that this lets a user of this category have access to the values stored by the attributes mentioned, if they are at two nodes unrelated under the `child`-axis. The policy for *Acad* is modeled as the negative clause (where the scope constraint is a conjunction of two scope constraints):

(2): $\leftarrow Acad(\text{id}),\ \text{Student.name},\ \text{Course.grade}$

$$[\text{Student.name } \texttt{child} \text{ Course.grade, Course.resp} \neq \text{id})].$$

The policy for *Stud* is modeled as the following clause, where '$*$' stands for *any* attribute:

(3): $\leftarrow Stud(\text{id}),\ \text{Student}.*\ [\text{Student.name} \neq \text{id}]$.

Here is a quick view on how such a clause-based vision can capture access control. We assume to start with (as is standard, cf. e.g., [7]), that the access to data stored at the nodes of the document is overseen by some central authority which has implemented the given access control policy; it is part of the assumption that this authority stores the history of the consultations of the document, by any given user, as a record representing the current state of the knowledge of the user, concerning data stored by the attributes referenced in the policy. (The authority can reinitialize these records at any moment of its choice.) The current state of data stored at a node $v$, that has been disclosed directly or indirectly, to a given user (with identifier *id*) having launched a query $Q$, will be represented as a set $Hist_{id}(v, Q)$ of pure positive clauses, containing at least the (positive) clause $Cat(id) \leftarrow$, where $Cat$ stands for the category of the user.

Assume then that a person of the category *Adm*, with identifier *id*, launches a query $Q$ on the document "Faculty", to collect all available information concerning the students of the faculty. We assume too, for the moment, having constructed the transition graph $G_Q$ associated with $Q$, and defined its runs evaluating $Q$ on $t$ (cf. Section 4). Suppose that we are at some some context node $u$ on $t$, under such a run, and a certain node $v$ is reached under the next transition of $G_Q$. This transition will correspond to a well-determined location step $Q'$ of $Q$; a set $Deduce_{id}(v, Q')$ of positive clauses will then be constructed too, to model the knowledge that the user can acquire – directly or indirectly – on the data stored at $v$ on $t$, by firing this transition. Then, by definition, the evaluating run of $G_Q$ on $t$ will select the node $v$, if and only if a condition of the following form is satisfied:

$$Deduce_{id}(v, Q') \cup Hist_{id}(v, Q) \cup \{(1)\} \not\models \bot$$

where (1) is the negative clause above, modeling the access control policy for the category *Adm*. Concretely, this will mean for the current example, that a person of the category *Adm* with identifier *id*, will not be able to access the grade obtained by the student concerned in any course, if the name of the student has already been accessed, under the current query $Q$ or some earlier query, launched by the same user, the data accessed under which is stored in $Hist_{id}(v, Q)$. The positive clauses of $Deduce_{id}(v, Q')$ will all be added to the history record of the user, for controlling the data (s)he accesses under future queries.

The construction of the sets of clauses – such as the set $Hist_{id}()$ above – needed for such a clause-based, step-wise evaluation of queries, constitutes the essence of Section 4. Section 2 presents our notation and terminology. Section 3 defines the transition graph $G_Q$, associated in a very simple manner with any given query $Q$; its runs are defined in terms of clauses, and are controlled by suitably formulated contraints; a technique for optimizing the step-wise run of $G_Q$ on any document $t$ is also defined, and it is shown that such an evaluation of $Q$ on $t$ is of linear complexity. Section 5 is devoted to showing that the clausal view is expressive enough. The concluding section presents some related work.

## 2 Preliminaries

We assume the reader to be familiar with the notion of XPath expressions, e.g., as defined in [12]. Our concern in this paper will be with XPath queries, i.e., queries involving not only the navigational axes of XPath, but also the data at the nodes of the documents (assumed given as unranked trees); the queries are assumed *positive*, in the sense that no negation is allowed on the navigational axes composing the query (but is allowed on the data filter parts of the query). For navigating on XML documents, the following XPath axes will be used: `self`, `child`, `parent`, `ancestor`, `descendant`, `following-sibling`, `preceding-sibling`; they will be referred to as *basic* in the sequel.

An alphabet $\Sigma$ is assumed given for naming the nodes (or tags) on the documents; these names will be referred to as nodenames or tagnames. *Att* will stand for the set of all attribute names at the nodes of all possible documents; *att* will stand for a variable running over *Att*; *val* will stand for any value assignable to an attribute, and $\mathcal{D}$ will denote the domain of all values for the attributes at the nodes on documents. A set of operators $Op = \{=, \neq, >, <, \geq, \leq\}$ is assumed given too, and *op* will stand for any operator in *Op*. We shall agree that the PCDATA at the nodes of the documents are in the format "*text* = data" where *text* is a special attribute (e.g., as in [10]).

A *filter type* is an expression generated by the non-terminal $F$ in the following grammar – where `A` stands for a basic XPath axis among the seven mentioned above, and $\sigma \in \Sigma \cup \{*\}$ ('$*$' stands here for an 'arbitrary' element of $\Sigma$):

$L ::= @\,att\ op\ \text{`}val'\ \mid\ position() = i\ \mid\ true$
$S ::= \texttt{A::}\sigma\ \mid\ \texttt{A::}\sigma\texttt{[}L\texttt{]}\ \mid\ \texttt{A::}\sigma\texttt{[}L\texttt{][}S\texttt{]}$
$G ::= S\ \mid\ \texttt{A::}\sigma\texttt{[}L\texttt{][}G\texttt{]}\ \mid\ G\ or\ G\ \mid\ G\ and\ G$
$F ::= L\ \mid\ G$

The expressions of the form $\texttt{[}F\texttt{]}$ where $F$ is a filter type, are said to be *filter expressions* or just *filters*. A filter expression of the form $\texttt{[}L\texttt{]}$, with no navigational axes, will be said to be *local*. We agree to identify $\texttt{A::}\sigma\texttt{[}true\texttt{]}$ with $\texttt{A::}\sigma$. Given a context node $u$ on any XML document $t$, any given filter expression $\texttt{[}F\texttt{]}$ evaluates to 'true' or to 'false' at $u$ on $t$; cf. e.g., [12].

Positive XPath *query expressions* are then defined as the expressions $Q$ generated by the grammar below – where `A`, $\sigma$ are as above, and $\texttt{[}F\texttt{]}$ is any filter expression:

$Q_0 ::= \texttt{/A::}\sigma\ \mid\ \texttt{/A::}\sigma\texttt{[}F\texttt{]}\ \mid Q_0\, Q_0$
$Q\ ::= Q_0\ \mid Q_0\texttt{/attribute::}att\ \mid\ Q_0\texttt{/attribute::}*\ \mid\ Q\ or\ Q$

(Note: `attribute` is a non-navigational axis of XPath, and $/@\,att$ is short for the syntax $\texttt{/attribute::}att$; the '$*$' in the last production stands for 'any'.) The query expressions generated by this grammar will be referred to as *canonical*. They are of the form $\texttt{/}C_1\texttt{/}\dots\texttt{/}C_n$, where each $C_i$ is either of the form $\texttt{A::}\sigma\texttt{[}F\texttt{]}$, or is a disjunction of the form $\texttt{A::}\sigma\texttt{[}F\texttt{]}\ or\ \texttt{A'::}\sigma'\texttt{[}F'\texttt{]}$. The filter expressions in the $C_i$ are referred to as their *filter components*. All our queries in the sequel will be assumed to be in canonical form. A query of the form $\texttt{/axis::}\sigma\texttt{[}F\texttt{]}$ will be said to be *elementary*.

A *location expression* (resp. *location step*) is defined as an expression of the form [axis::$\sigma$[$F$]] (resp. axis::$\sigma$[$F$]) where axis is one of the seven (basic) navigational axes mentioned above, $\sigma \in \Sigma$, and [$F$] is a filter. If the filter is local, i.e., *without* navigational axes, the location expression (resp. location step) will be said to be *atomic*. Let $e =$ [axis::$\sigma$[$F$]] be any given location expression, and $u$ a given context node on an XML document $t$. We shall say that $e$ evaluates to 'true' at $u$ on $t$, if and only if there is some node $v$ on $t$ such that the relation $u$ axis $v$ holds on $t$ (i.e., $v$ is an axis-node wrt $u$, on $t$), the tagname at $v$ is $\sigma$, and the filter [$F$] evaluates to true at $v$.

Given a document $t$, two nodes $u, v$ on $t$, a query of the form /axis::$\sigma$[$F$], we say that $v$ is *selected* by this query, wrt the context node $u$, iff $u$ axis $v$ holds on $t$, the tagname at $v$ is $\sigma$, and the filter [$F$] evaluates to true at $v$. The notion of selection of any given node $v$, on a document $t$, wrt the *Root* node of $t$, by any given query expression $Q$, is then defined inductively. (By *Root*, we mean the fictive root node that is assigned in XML, to any document $t$, just above the actual root node of the tree $t$ which will be denoted as *root*.) It will be convenient to write any location step in the form axis::$\sigma$[$L$][$F$], where [$L$] is local, and the navigational axes are all in [$F$].

For any context node $u$ on any document $t$, let $Data_t(u)$ denote the data stored at $u$ on $t$, i.e., the set of all equalities ($att = val$) stored at $u$; we then define $t(u)$ as the pair $(\sigma, Data_t(u))$, where $\sigma$ is the nodename of $t$ at $u$. If $Q =$/$C_1$/.../$C_n$ is any query in canonical form, the *answer* for $Q$ on $t$ is the set $Eval_t(Q)$ of all $t(u)|_Q$, where $u$ runs over the set of nodes selected by $Q$ on $t$, and $t(u)|_Q$ is the projection of $t(u)$ on the set of attributes selected by $C_n$. Two queries $Q_1, Q_2$ are said to be *equivalent* (in notation $Q_1 \equiv Q_2$) iff, on any document $t$, the answer sets for $Q_1$ and $Q_2$ are the same.

**Remark**. a) The axes following and preceding of XPath have not been considered above, since we have the following equivalences:
/following::$\sigma$ $\equiv$ /ancestor-or-self::$*$/following-sibling::$*$
/descendant-or-self::$\sigma$,
/preceding::$\sigma$ $\equiv$ /ancestor-or-self::$*$/preceding-sibling::$*$
/descendant-or-self::$\sigma$.
b) We shall also need two further navigational axes that are only implicitly defined in XPath, namely the right-sibling and left-sibling axes; we shall refer to these as right,left, respectively; by definition, we have (cf. [12]):
right::$\sigma$ is equivalent to
following-sibling::$\sigma$[$position() = 1$],
left::$\sigma$ is equivalent to
preceding-sibling::$\sigma$[$position() = 1$].
c) To each axis among the seven basic axes above, we associate a step-wise axis denoted as dir-axis, the role of which is to move from node to node (on any tree $t$), along the direction given by axis. Here is the correspondence table:

| axis | dir-axis |
|---|---|
| `self` | `self` |
| `child` | `child` |
| `descendant` | `child` |
| `parent` | `parent` |
| `ancestor` | `parent` |
| `following-sibling` | `right` |
| `preceding-sibling` | `left` |

d) To any given basic XPath axis `axis`, we associate a (Boolean valued) unary predicate `Fin-axis()`, which evaluates to 'true' at any node $u$ on any given XML document $t$, iff there exists no node $v$ such that $u$ `dir-axis` $v$ holds on $t$. (We shall agree that `Fin-self()` always evalutes to 'true'.) For an `axis` $\neq$ `self` it evaluates to 'true' at the *root*, or at the leafs, or at the right-most or left-most nodes on $t$.

We propose to evaluate any (positive XPath) query $Q$ on any document $t$ in the following manner: $Q$ will be seen as a concatenation of its sub-expressions of certain specific types, to each of which we shall associate a transition graph; the transition graph $G_Q$ associated with $Q$ is defined as the join of all these individual transition graphs, in a certain sense. The evaluation of $Q$ on any given document $t$ is then defined via a notion of *step-wise run* $G_Q$ on $t$; each such run is formulated as a (propositional) clause. We show that, under such a vision, the run of $G_Q$ evaluating $Q$ on $t$ can be controlled by a linear optimization strategy; one deduces thus a complexity bound for the evaluation, which is linear on the number of atomic location steps in $Q$, and the number of edges on $t$.

Such a vision for the evaluation of any query $Q$, in terms of a step-wise run of $G_Q$ on $t$, allows us also to integrate access control policies on the given document $t$, into the evaluation mechanism: the conditions of access to data, as stipulated by the policies, are also expressed as a set of first-order clauses over the attribute names and/or attribute values, possibly subject to some additional *scope constraints*; these clauses will be referred to as *access control clauses*; the run of $G_Q$, evaluating the query $Q$ on $t$, will then be controlled by clausal resolution (cf Section 4).

## 3 Transition Graph for a Query

We present here our evaluation formalism based on transition graphs, without considering any access control policy. We saw above that every canonical query $Q$ is of the form $Q = /C_1/C_2/\ldots/C_n$, where for every $1 \leq i \leq n$, the elementary subquery $/C_i = /\mathtt{axis}_i::\sigma_i[L_i][F_i]$ contains a navigational part $/\mathtt{axis}_i::\sigma_i[Li]$ with local filter $[L_i]$, and a filter part $[F_i]$. The transition graph $G_Q$, that evaluates a query $Q$ on any given document $t$, will be defined as a concatenation of deterministic *elementary transition graphs* $G_i$, for $1 \leq i \leq n$, associated with the elementary query $/C_i$. The role of $G_i$ is to find all nodes of $t$, answering the query $/C_1/\ldots/C_i$. This will be done with the help of the run of $G_i$ on $t$. Starting from *Root* if $i = 1$, and from every node answering the query /$C_1$/.../$C_{i-1}$, if $i > 1$, the run of $G_i$ will progress along a path specified by $\texttt{axis}_i$, and select the nodes where $C_i$ evaluates to 'true'. (Note: *Root* is the fictive root assigned in XML to any document $t$, just above the actual root node of the tree $t$.)

For redactional simplicity, we shall assume that the filter components of $Q$ are free from conjunction and disjunction (the developments given below are easily extended to cover such cases). In subsection 3.1, we give the construction of the graph $G_i$ for an atomic elementary query /$C_i$, i.e., an elementary query where the unique filter component is local; in subsection 3.2, we construct the transition graph for an elementary query with a non-local filter; and we finally construct, in subsection 3.3, the transition graph $G_Q$ for the entire query $Q$; we also present an optimizaition strategy, and prove that under this strategy, our evaluation approach is linear with respect to the number of atomic location steps composing $Q$ and the number of edges of the document. The graphs look all very much alike, but their runs do not have the same semantics. An illustrative example is given in subsection 3.4.

### 3.1 ETG for an atomic elementary query

Every query /$C_i$ considered in this subsection is an atomic elementary query of the form /$C_i$ =/$\texttt{axis}_i$::$\sigma_i$[$L_i$], where $\texttt{axis}_i$ is basic, $\sigma_i \in \Sigma \cup \{*\}$, and the filter [$L_i$] is local.

**Definition 1.** *An elementary transition graph (ETG) for the atomic elementary query $C_i$ =/$\texttt{axis}_i$::$\sigma_i$[$L_i$], is the graph $G_i = (States_i, Trans_i)$, over the alphabet $\{x, y\}$, with $x = \sigma_i$[$L_i$], and $y = \overline{\sigma_i\texttt{[}L_i\texttt{]}}$, represented in Figure 1, where $init_i$ is the initial state, and $ok_i$ is the unique selecting state.*

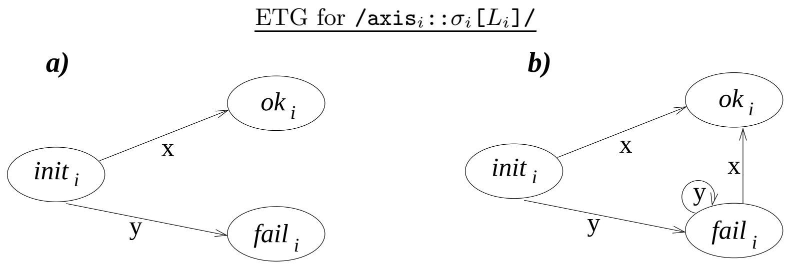



**Fig. 1.** ETG $G_i$ for /$C_i$ = $\texttt{axis}_i$::$\sigma_i$ [$L_i$], where: **a)** $\texttt{axis}_i \in$ {self, child, parent}; **b)** $\texttt{axis}_i \notin$ {self, child, parent}

(Note: The $y = \overline{\sigma_i\texttt{[}L_i\texttt{]}}$, in the definition above, stands for the complement of the 'data' represented by $\sigma_i$[$L_i$].) Consider a canonical query $Q$ =/$C_1$/.../$C_n$, and a document $t$. The role of the graph $G_i$ for /$C_i$, is to select the nodes of $t$ answering /$C_1$/.../$C_i$; this is done by a run of $G_i$ on $t$. If the ETG

$G_i$ corresponds to $/C_i = \texttt{axis}_i\texttt{::}\sigma_i\texttt{[}L_i\texttt{]}$, the *run* of $G_i$ on $t$ is a mapping $M_i\colon Nodes(t) \to \mathcal{P}(States_i)$, satisfying the following clausal conditions, where $u, v$ are nodes on $t$, $\langle q, u\rangle$ stands for $q \in M_i(u)$, and $z$ stands for $x$ or $y$:

1$a$. $\langle init_i, u\rangle \leftarrow u = Root$, if $i = 1$;
$\langle init_i, u\rangle \leftarrow \langle ok_{i-1}, u\rangle$, if $i > 1$;
2$a$. $\langle q', v\rangle \leftarrow \langle q, u\rangle$, if $(t(v) \models z)$, $(u\,\texttt{dir-axis}_i\,v)$, and $q \xrightarrow{z} q' \in Trans_i$;
3$a$. $\langle init_i, u\rangle \leftarrow \langle ok_i, u\rangle$, if $\texttt{axis}_i \notin \{\texttt{self, child, parent}\}$.

*Semantics*: The set of states $M_i(u)$ assigned to $u$ will actually be constructed, incrementally, under a step-wise traversal of $t$. The run $M_i$ of the ETG $G_i$ on $t$ starts (clause 1$a$) by assigning the initial state $init_i$ to the nodes $v_{i-1}$ answering the subquery $/C_1/\ldots/C_{i-1}$ (the run $M_1$ starts at the fictive root node *Root* of the tree $t$). Then $M_i$ progresses along the path defined by $\texttt{axis}_i$ (clause 2$a$); it assigns the state $fail_i$ to every node $w$, such that $\sigma_i\texttt{[}L_i\texttt{]}$ is not valid under $t(w)$, and the selecting state $ok_i$ to the nodes $v$ where $\sigma_i\texttt{[}L_i\texttt{]}$ is valid under $t(v)$. When $\texttt{axis}_i \notin \{\texttt{self, child, parent}\}$, the run $M_i$ will continue, using the clause 3$a$ for $u = v$, assign the initial state $init_i$ to the already selected node $v$, and look for other nodes where the data validates $\sigma_i\texttt{[}L_i\texttt{]}$.

### 3.2 ETG for a non-atomic elementary query

The elementary query $/C_i$, composing $Q$, of the form $/C_i = /\texttt{axis}_i^0\texttt{::}\sigma_i^0\texttt{[}L_i^0\texttt{]}\texttt{[}F_i\texttt{]}$, can be wriiten as $/C_i = /step_i^0\texttt{[}step_i^1\texttt{[}step_i^2\texttt{[}\ldots\texttt{[}step_i^{k(i)}\texttt{]}\ldots\texttt{]}$, for some positive integer $k(i)$, where $step_i^p = \texttt{axis}_i^p\texttt{::}\sigma_i^p\texttt{[}L_i^p\texttt{]}$ is an atomic location step, for every $0 \leq p \leq k(i)$. The elementary transition graph $G_i$ for such an elementary subquery $/C_i$, will be defined as a concatenation of what we shall call *one-step transition graphs* (STG, for short) $G_i^p$, where each STG $G_i^p$ corresponds to an atomic location step $step_i^p$ appearing in $C_i$. We start by giving the construction of the STG $G_i^p = (States_i^p, Trans_i^p)$, which depends on the role played by the location step $step_i^p = \texttt{axis}_i^p\texttt{::}\sigma_i^p\texttt{[}L_i^p\texttt{]}$. In Figure 2, 3, 4, for every $p \in \{0, \ldots, k(i)\}$, $x$ stands for $\sigma_i^p\texttt{[}L_i^p\texttt{]}$, and $y$ for $\overline{\sigma_i^p\texttt{[}L_i^p\texttt{]}}$. We have three types of STG:
The STG $G_i^0$, corresponding to $/step_i^0\texttt{[}$ (navigational part followed by a non-local filter), is represented in Figure 2.
The STG $G_i^p$, corresponding to $\texttt{[}step_i^p\texttt{[}$, $p \in \{1, \ldots, k(i) - 1\}$ (a non-local filter followed by a non-local filter), is given in Figure 3.
The STG $G_i^{k(i)}$, corresponding to $\texttt{[}step_i^{k(i)}\texttt{]}$ (the last non-local filter in $/C_i$), is given in Figure 4.
The role of the STG $G_i^0$ is to find the nodes $v$ *potentially* selected by $/C_1/\ldots/C_i$ (i.e., $v$ such that $t(v) \models \sigma_i^0\texttt{[}L_i^0\texttt{]}$); the graphs $G_i^p$, for $1 \leq p \leq k(i)$, are used to *validate* the selections of the graph $G_i^0$ (i.e., they retain only the potentially selected nodes, where $\texttt{[}F_i\texttt{]}$ evaluates to 'true'). Having constructed the one-step transition graphs $G_i^p$, we can now define the transition graph $G_i$ for the elementary query with a non-local filter component.

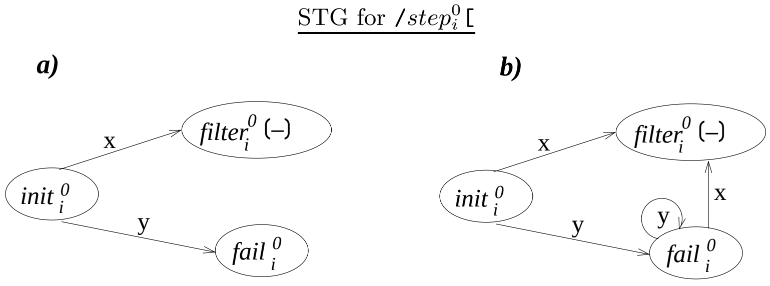



**Fig. 2.** STG $G_i^0$ for `axis`$_i^0$`::`$\sigma_i^0$ `[`$L_i^0$`][ ]`, where: **a)** `axis`$_i^0 \in \{$`self, child, parent`$\}$; **b)** `axis`$_i^0 \in \{$`descendant`, `ancestor`, `following-sibling`, `preceding-sibling`$\}$

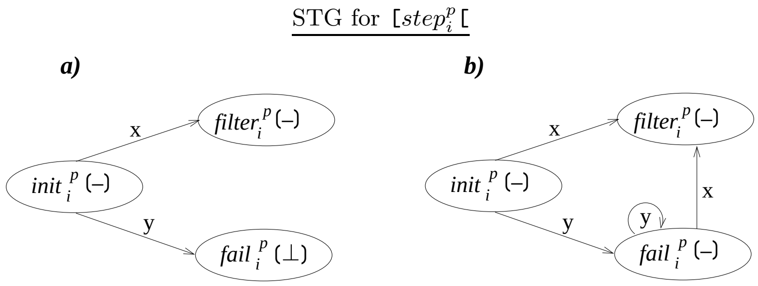



**Fig. 3.** STG for `axis`$_i^p$`::`$\sigma_i^p$ `[`$L_i^p$`]`, where: **a)** `axis`$_i^p \in \{$`self, child, parent`$\}$; **b)** `axis`$_i^p \in \{$`descendant, ancestor, following-sibling, preceding-sibling`$\}$

**Definition 2.** *The ETG $G_i$, for* $/C_i = /step_i^0[step_i^1[step_i^2[\ldots[step_i^{k(i)}]\ldots]$*, with a non-local filter component, is a graph $G_i = (States_i, Trans_i)$, where:*

- $States_i = \bigcup_{p=0}^{k(i)} States_i^p \bigcup_{p=0}^{k(i)-1} ok_i^p[\alpha] \bigcup_{p=1}^{k(i)} \{q_i^p[\alpha] \mid q \in \{init, fail\}\}$,
- $Trans_i = \bigcup_{p=0}^{k(i)} Trans_i^p$,

*where $\alpha$ stands for $\bot$ or $\top$, $init_i^0$ is the initial state, and $ok_i$ is the unique selecting state.*

Given the ETG $G_i$, and a document $t$, the *run* of $G_i$ on $t$, is a mapping $M_i\colon Nodes(t) \to \mathcal{P}(States_i)$, satisfying the clausal conditions $1-12$ given below. We first introduce some notation and terminology that will be used in the formulation of these clauses. We consider a 3-valued logic over the truth table $\{1, 0, \omega\}$ (where $\omega$ means 'undefined'), the truth values are ordered as $0 < \omega < 1$, and we set: $\overline{1} = 0, \overline{0} = 1, \overline{\omega} = \omega$.

• Let $i, 1 \leq i \leq n$, and $p, 1 \leq p \leq k(i)$, be given, and let `axis`$_i^p$ be the (basic) axis of the corresponding location step in $Q$. Then, for any given document $t$, we

ETG for [$step_i^{k(i)}$]

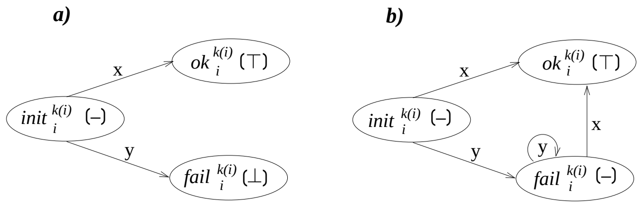



**Fig. 4.** STG for [$\texttt{axis}_i^{k(i)}$::$\sigma_i^{k(i)}$ [$L_i^{k(i)}$]; **a)** $\texttt{axis}_i^{k(i)} \in \{\texttt{self, child, parent}\}$; **b)** $\texttt{axis}_i^{k(i)} \in \{\texttt{descendant, ancestor, following-sibling, preceding-sibling}\}$

define a (3-valued) unary predicate $\Omega_i^p$ on $t$, to be evaluated at any given node $u$ on $t$, recursively as follows:

i) Case where $\texttt{Fin-axis}_i^p(u)$ is true:
$\Omega_i^p(u) = 1$ iff a state of the form $q_i^p[\top]$ is in $M_i(u)$;
$\Omega_i^p(u) = 0$ iff $M_i(u)$ has a state of the form $q_i^p[\bot]$;
$\Omega_i^p(u) = \omega$ otherwise.
ii) Case where $\texttt{Fin-axis}_i^p(u)$ is false:
$\Omega_i^p(u) = Sup_v\{\Omega_i^p(u) \mid u \ \texttt{dir-axis}_i^p\ v\}$

The Transition Rules of $M_i$
(Notation: $u, v$ are nodes on $t$, $p \in \{0, \ldots, k(i)\}$, $q, q' \in \{init, ok, fail\}$; for any state $s$, $\langle s, u\rangle$ stands for $s \in M_i(u)$; $z$ is $x$ or $y$; the sematics of each rule is given in parentheses.)

1. $\langle init_i^0, u\rangle \leftarrow u = Root$, if $i = 1$;
$\langle init_i^0, u\rangle \leftarrow \langle ok_{i-1}, u\rangle$, if $i > 1$
(rule to go from $G_{i-1}$ to $G_i$)
2. $\langle q', v\rangle \leftarrow \langle q, u\rangle$,
if $(t(v) \models z)$, $(u\,\texttt{dir-axis}_i\,v)$, and $q \xrightarrow{z} q' \in Trans_i$
(transition from node $u$ to node $v$)
3. $\langle init_i^{p+1}[-], u\rangle \leftarrow \langle ok_i^p[-], u\rangle$, if $p \leq k(i) - 1$
(rule to go from $G_i^p$ to $G_i^{p+1}$)

Validate the selection

4. for $p \geq 1$: $\langle q_i^p[\top], u\rangle \leftarrow \langle q_i^p[-], u\rangle$, $\Omega_i^p(u)$
(back-propagate $[\top]$ along path traversed by $G_i^p$)
5. $\langle ok_i^{p-1}[\top], u\rangle \leftarrow \langle ok_i^{p-1}[-], u\rangle$, $\langle init_i^p[\top], u\rangle$
($step_i^p$ signals filter validation to $step_i^{p-1}$)
6. $\langle ok_i, u\rangle \leftarrow \langle ok_i^0[\top], u\rangle$
(validate potential selection at $u$)
7. $\langle init_i^0, u\rangle \leftarrow \langle ok_i, u\rangle$,
if $\texttt{axis}_i \notin \{\texttt{self, child, parent}\}$
(continue with $M_i$ from a validated node $u$)

Invalidate the selection

8. $\langle fail_i^p[\bot], u\rangle \leftarrow \langle fail_i^p[-], u\rangle$, $\texttt{Fin-axis}_i^p(u)$, if $p \geq 1$
   (we are at an 'end', $step_i^p$ is false)
9. for $p \geq 1$: $\langle q_i^p[\bot], u\rangle \leftarrow \langle q_i^p[-], u\rangle$, $\overline{\Omega_i^p(u)}$
   (back-propagate $[\bot]$ along path traversed by $G_i^p$)
10. $\langle ok_i^{p-1}[\bot], u\rangle \leftarrow \langle ok_i^{p-1}[-], u\rangle$, $\langle init_i^p[\bot], u\rangle$,
    if $\texttt{axis}_i^{p-1} \in \{\texttt{self, child, parent}\}$
    ($step_i^{p-1}$ is false at node $u$)
11. $\langle init_i^{p-1}[-], u\rangle \leftarrow \langle ok_i^{p-1}[-], u\rangle$, $\langle init_i^p[\bot], u\rangle$,
    if $\texttt{axis}_i^{p-1} \notin \{\texttt{self, child, parent}\}$
    (filter false at $u$ for $step_i^p$, continue with $step_i^{p-1}$)
12. $\langle init_i^0, u\rangle \leftarrow \langle ok_i^0[\bot], u\rangle$,
    if $\texttt{axis}_i \notin \{\texttt{self, child, parent}\}$
    (continue with $M_i$ from an invalidated node $u$).

*Semantics*: The role of the run $M_i$ of $G_i$ on $t$, is to select every node $u$ of $t$, where $\sigma_i^0\texttt{[}L_i^0\texttt{]}$ is valid under the data $t(u)$, and the filter component $\texttt{[}F_i\texttt{]}$ evaluates to 'true'. $M_i$ starts by assigning the state $init_i^0$, to the nodes $v_{i-1}$ answering the subquery $\texttt{/}C_1\texttt{/}\ldots\texttt{/}C_{i-1}$ (clause 1). Next, using the transitions of $G_i^0$, it progresses along the path defined by $\texttt{axis}_i^0$; it assigns the state $fail_i^0$ to the nodes where the data does not validate $\sigma_i^0\texttt{[}L_i^0\texttt{]}$, and the state $ok_i^0[-]$ to the first node $v_i^0$, such that $\sigma_i^0\texttt{[}L_i^0\texttt{]}$ is valid under $t(v_i^0)$ (clause 2). Every such node $v_i^0$ is potentially selected; it will get ultimately selected by $M_i$ iff the filter component $\texttt{[}F_i\texttt{]}$ evaluates to 'true' at $v_i^0$. To check the satisfaction of the filter at $v_i^0$, the transitions of $G_i^p$, for $p \in \{1,\ldots,k(i)\}$ are used. For any $p \in \{0,\ldots,k(i)-1\}$, let $v_i^p$ denote the node to which $M_i$ has assigned the state $ok_i^p[-]$. In this case, clause 3 will be used (with $u = v_i^p$), and the state $init_i^{p+1}$ will be assigned to $v_i^p$. The run $M_i$ will continue along the path defined by the $\texttt{axis}_i^{p+1}$; under the transitions of $G_i^{p+1}$, it will assign:

- $fail_i^{p+1}[-]$ to every node where the data does not validate $\sigma_i^{p+1}\texttt{[}L_i^{p+1}\texttt{]}$,
- if $p+1 < k(i)$, the state $ok_i^{p+1}[-]$ to the first node where the data validates $\sigma_i^{p+1}\texttt{[}L_i^{p+1}\texttt{]}$,
- if $p+1 = k(i)$, the state $ok_i^{k(i)}[\top]$ to the first node where the data validates $\sigma_i^{k(i)}\texttt{[}L_i^{k(i)}\texttt{]}$.

If, for some node $v_i^p$, $0 \leq p \leq k(i)-1$, such that $\texttt{axis}_i^p \notin \{\texttt{self, child, parent}\}$, and the data at *any* node $w$ satisfying $v_i^p\ \texttt{axis}_i^{p+1}\ w$, on $t$, does not validate $\sigma_i^{p+1}\texttt{[}L_i^{p+1}\texttt{]}$, this means the filter $\texttt{[}step_i^{p+1}\texttt{]}$ is not satisfied at $v_i^p$. In this case, the clauses 8, 9 and 11 (where $p := p+1$) will be used, and $init_i^p[-]$ will get assigned to $v_i^p$; $M_i$ will then search for other possible nodes, where the data validates $\sigma_i^p\texttt{[}L_i^p\texttt{]}$. (Note that just as the set $M_i(u)$, of states assigned to any given node $u$ on $t$, evolves when $M_i$ traverses $t$, the truth value of the predicate $\Omega_i^p(u)$ will also evolve along the run; note also that only the truth values 1 and 0 for this predicate appear in the transitions.)

**Validate potential selection at $v_i^0$:** If the state $ok_i^{k(i)}[\top]$ is reached under $M_i$, the rules 4, 5 will be used, and for any node $u$ traversed by $M_i$ with a state of the form $s[-]$ in $M_i(u)$, the state $s[\top]$ will also get added to $M_i(u)$. In particular $ok_i^0[\top]$ will get added to $M_i(v_i^0)$; and by rule 6 the selecting state $ok_i$ will also get added to $M_i(v_i^0)$.

**Invalidate potential selection at $v_i^0$:** The potential selection at $v_i^0$ is invalidated if the filter `[`$F_i$`]` evaluates to 'false' at $v_i^0$. In this case, we propagate the invalidation symbol $[\bot]$ to all nodes traversed by $M_i$, by using the rules 8-10. In particular, the state $ok_i^0[\bot]$ will get added to $M_i(v_i^0)$, and thus this node could not be selected.

**Continue the run from $v_i^0$:** if $\texttt{axis}_i^0 \in \{$`descendant, ancestor, following-sibling, preceding-sibling`$\}$, the run $M_i$ will continue, from a validated or invalidated potentially selected node $v_i^0$, by using rule 7 or rule 12.

### 3.3 The Transition Graph for Q

We define here the transition graph $G_Q$ for the entire canonical query $Q$.

**Definition 3.** *Given a query $Q =$`/`$C_1$`/`$\ldots$`/`$C_n$, and the ETGs $G_i$ associated with the queries `/`$C_i$, $i \in \{1, \ldots, n\}$, the transition graph (TG) evaluating $Q$ is the graph $G_Q = (States_Q, Trans_Q)$, where $States_Q = \bigcup_{i=1}^n States_i$, $Trans_Q = \bigcup_{i=1}^n Trans_i$, the initial state $init_Q$ is defined as $init_1$ (or $init_1^0$), and the unique selecting state is $ok_n$.*

Given any document $t$, and the runs $M_i$ of the ETG $G_i$ on $t$, for $1 \leq i \leq n$, the *run* of $G_Q$ on $t$ is defined as a mapping $M\colon Nodes(t) \to \mathcal{P}(States_Q)$, with $M = \bigcup_{i=1}^n M_i$, and satisfying the following two additional clausal conditions, to continue the search, from a selected node $u$, for other nodes possibly answering $Q$ (depending on whether the filter in `/`$C_1$ is local or not):

13. $\langle init_1, u\rangle \leftarrow \langle ok_n, u\rangle$, if $\texttt{axis}_1 \notin \{\texttt{self}, \texttt{child}, \texttt{parent}\}$
14. $\langle init_1^0, u\rangle \leftarrow \langle ok_n, u\rangle$, if $\texttt{axis}_1^0 \notin \{\texttt{self}, \texttt{child}, \texttt{parent}\}$

The answer for $Q$ on $t$ is the set of all nodes $u$ of $t$, such that $ok_n \in M(u)$.

**An Optimizing Strategy** Just like for ETGs or STGs, a run $M$ of the graph $G_Q$, for the query $Q$ on the document $t$, constructs the set $M(u) \subset States_Q$ at any node $u$, incrementally, in a step-wise fashion; $M$ starts at the *Root* and traverses $t$ along the various axes of the succesive location steps in $Q$. By a transition, or "move", of $G_Q$ from any given node $u$, we shall mean a resolution step from a positive clause $\langle q, u\rangle \leftarrow$, into one of the clauses $1a$-$3a$ or 1-14 defining $M$, given above. Note: such a move of $G_Q$ may not always be from $u$ to a different node $v$ on $t$: it may just add a further state to the set $M(u)$.

The run $M$ can actually be controlled with a strategy – referred to as the *linear strategy*– to avoid redundant transitions, and reduce the complexity of the evaluating run of $G_Q$. It is based on the simple idea that at any node $u$, the satisfaction of $step_i^p$ (resp. $step_i$ in atomic case) need not be checked more than once; it is formalized as follows:

• At any given instant $\tau$ of the step-wise run $M$ of $G_Q$ on $t$, and to any given edge $\mathbf{e}$ on the document $t$, let $G_i^p(\mathbf{e}, \tau)$ be the set of the transition rules of the graph $G_i^p$ that have already been used by $M$ (up to instant $\tau$), between the two nodes connected by $\mathbf{e}$; then no future transition (for $\tau'$ later than $\tau$) between these two nodes may employ any of the rules already present in the set $G_i^p(\mathbf{e}, \tau)$. Note: since transitions could be from a node to itself, we shall agree that $t$ has "fictive" edges from any node $u$ to itself, for this strategy, We then deduce the following proposition:

**Proposition 1.** *The complexity of the step-wise evaluation of any XPath query $Q$ on any XML document $t$ by the run of $G_Q$, under the above strategy, is linear on the number of atomic location steps in $Q$ and the number of edges on $t$.*

*Proof.* No edge on $t$ is traversed more than once by any transition of any of the graphs $G_i^p$; on the other hand, for any given edge on $t$, checking if a transition rule of any given $G_i^p$ has already served along that edge, is done in linear time wrt the number of location steps composing the query $Q$. □

### 3.4 Example

We evaluate here the query $Q = \texttt{/descendant::}a\texttt{[ancestor::}b\texttt{]/parent::}c$, on the tree $t$ represented in Figure 5. The nodes of $t$ are denoted by their positions.

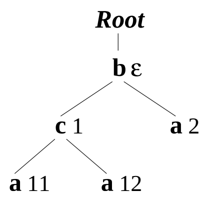



**Fig. 5.** XML document $t$

Figure 6, presents the part of the run of the automaton $G_Q$, evaluating the first elementary query $/C_1 = \texttt{/descendant::}a\texttt{[ancestor::}b\texttt{]}$. We first consider the navigational part $step_1^0 = \texttt{descendant::}a$. Starting from the fictive *Root* of $t$, the run traverses the document top-down (`descendant` axis), using the clauses:

– $\langle init_1^0, Root\rangle \leftarrow$
– $\langle fail_1^0, \varepsilon\rangle \leftarrow \langle init_1^0, Root\rangle$
– $\langle fail_1^0, 1\rangle \leftarrow \langle fail_1^0, \varepsilon\rangle$
– $\langle ok_1^0[-], 2\rangle \leftarrow \langle fail_1^0, \varepsilon\rangle$
– $\langle ok_1^0[-], 11\rangle \leftarrow \langle fail_1^0, 1\rangle$
– $\langle ok_1^0[-], 12\rangle \leftarrow \langle fail_1^0, 1\rangle$

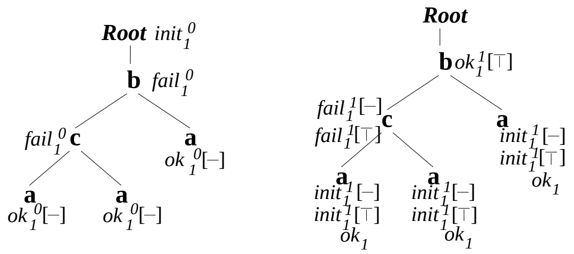



**Fig. 6.** Evaluation of `/descendant::`$a$`[ancestor::`$b$`]`

The states assigned to the nodes of $t$ are presented to the left of Figure 6. At the nodes $u = 2$, 11, 12, the run of $step_1^0$ uses the clause $\langle init_1^1[-], u\rangle \leftarrow \langle ok_1^0[-], u\rangle$, and starts its bottom-up move for evaluating the filter part $step_1^1 =$ `ancestor::`$b$ (Figure 6, to the right) . The following clauses are then used for the run of the STE $G_1^1$:

- $\langle fail_1^1[-], 1\rangle \leftarrow \langle init_1^1[-], 11\rangle$
- $\langle ok_1^1[\top], \varepsilon\rangle \leftarrow \langle fail_1^1[-], 1\rangle$
- $\langle fail_1^1[-], 1\rangle \leftarrow \langle init_1^1[-], 12\rangle$
- $\langle ok_1^1[\top], \varepsilon\rangle \leftarrow \langle init_1^1[-], 2\rangle$

Next, using the rule 4 from the definition of the run, the symbol $[\top]$ is back-propagated from the node $\varepsilon$ to the nodes 1, 2, and then from 1 to 11 and 12. Finally, with the help of the rules:

- $\langle ok_1, u\rangle \leftarrow \langle ok_1^0[\top], u\rangle$
- $\langle init_2, u\rangle \leftarrow \langle ok_1, u\rangle$

the state $ok_1$ is assigned to the nodes $u = 11$, 12, 2, which become the context nodes for the second elementary query $/C_2 =$ `/parent::`$c$.

The evaluation of $/C_2$ is presented in Figure 7. Starting from the context nodes 11 and 2, the run moves bottom-up; and using the clauses:

- $\langle ok_2, 1\rangle \leftarrow \langle init_2, 11\rangle$
- $\langle init_2, 1\rangle \leftarrow \langle ok_2, 1\rangle$
- $\langle fail_2, \varepsilon\rangle \leftarrow \langle init_2, 1\rangle$
- $\langle fail_2, \varepsilon\rangle \leftarrow \langle init_2, 2\rangle$

it selects the node 1, and refuses the selection of the node $\varepsilon$. Then it starts from the context node 12, and using the rule $\langle ok_2, 1\rangle \leftarrow \langle init_2, 12\rangle$, it moves to the node 1. By the linear strategy, we avoid using the clause $\langle fail_2, \varepsilon\rangle \leftarrow \langle init_2, 1\rangle$ for moving from node 1 to $\varepsilon$, for a second time.

Under this run, we have selected the node 1. (In general case, we should then use the clause $\langle init_i^0, 1\rangle \leftarrow \langle ok_2, 1\rangle$, to look for the other nodes possibly answering $Q$.)

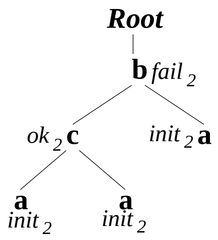



**Fig. 7.** Evaluation of .../`parent::`$c$

### 3.5 Evaluating runs interpreted differently

We propose here a look at the evaluating run of the transition graph $G_Q$, for any given canonical query $Q$ on any document $t$. from a slightly different viewpoint.

1) We first observe that several of the transitions – such as those back-propagating (or signaling) a filter validation or filter failure – can be seen as synchronization steps between transitions of a simpler form. It thus seems possible to define the transition graph for $Q$ as a Petri net $P_Q$, and its run evaluating $Q$ on a document $t$, as a synchronous product of the Petri Net $P_Q$, with $t$, in a sense to be formalized.

2) We observe next that any canonical query $Q = /C_1/\ldots/C_n$ can be seen as equivalent to a sequence of queries, each of which is in a *standard form* in the sense of [4]; and this can be done in such a way that the number of queries (in standard form) in the sequence is at most twice the number of atomic location steps composing $Q$. This is done as follows. To start with, we define, for every basic axis, an inverse:

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

With the notation of our earlier sections, for every $i, 1 \leq i \leq n$, and $p, 1 \leq p \leq k(i)$, we introduce then a symbol denoted as $ok_i$ and $ok_i^p$, and define $ok_0$ as *root*. Each of these symbols is to be assigned (as an additional label, or state), in a recursive manner, to the nodes on any given document $t$, which answer some specific queries in a standard form, as stipulated below:

Case i) $/C_i$ is atomic, of the form $/\texttt{axis}_i\texttt{::}\sigma_i\texttt{[}L_i\texttt{]}$:

In this case, $ok_i$ is a label (state) to be assigned to nodes answering the following query: $/\texttt{descendant::}\sigma_i\texttt{[axis}_i^{-1}\texttt{::}ok_{i-1}\texttt{]}$,

Case ii) $/C_i$ is non-atomic, of the form

$$/\texttt{axis}_i^0\texttt{::[}\sigma_i^0\texttt{[}L_i^0\texttt{][axis}_i^1\ \texttt{[}\sigma_i^1\texttt{[}L_i^1\texttt{][}\ldots\texttt{[axis}\ _i^{k(i)}\texttt{::[}\sigma_i^{k(i)}\texttt{[}L_i^{k(i)}\texttt{]]}\ldots\texttt{]:}$$

In this case, assign:

$ok_i^{k(i)}$ to nodes answering `/descendant::`$\sigma_i^0$`::[`$L_i^0$`][(axis`$_i^0$`)`$^{-1}$`::`$ok_{i-1}$`];`

for every $p, 1 \le p \le k(i) - 1$:

$ok_i^p$ to nodes answering `/descendant::`$ok_i^{p-1}$`[axis`$_i^p$`::`$\sigma_i^p$`[`$L_i^p$`]];`

and finally:

$ok_i$ to nodes answering `/descendant::`$ok_i^{k(i)-1}$`[axis`$_i^{k(i)}$`::`$\sigma_i^{k(i)}$`[`$L_i^{k(i)}$`]].`

Such a view of the run is of interest, since it will allow us to restrict to top-down runs of suitably defined automata; this will help us cover the case of queries on compressed documents as well.

## 4 Access Control: Definitions

In this section, we formalize the clause-based view for access control (that was illustrated in Section 1). The definitions are all in an intentionally light style.

*Definition (1):* By *User*, we shall mean a (finite) set of individual users, while by *Category* we shall mean a finite set of users, each such set with a given specific name, or label (as in the Example of Section 1). Both are seen as unary predicates: *User()*, *Cat()*. For defining the other notions needed, we assume given a document $t$, on which a given access control policy has been enforced. It is assumed that every condition of the policy controlling data access on $t$, is expressed as a pure negative clause in the format we saw earlier, possibly subject to a scope constraint; the set of all such negative (constrained) clauses will again be denoted as $\mathcal{P}$. Every literal appearing in any of these clauses is thus assumed to be either a unary predicate (of the form *User()* or *Cat()*), or a propositional symbol of the form $u.a$ where $u$ is a nodename, and $a$ is an attribute name. The *scope constraint* of a clause in $\mathcal{P}$ is, by definition, a constraint of the form [*constr*] where *constr* is a (finite) conjunction of expressions of the form $u_i.a_i$ `axis`$_i$ $v_i.b_i$ (where `axis`$_i$ is a basic axis or a corresponding `dir-axis`), and/or of expressions of the form $u_j.a_j$ *op val*, where *val* is a data value in $\mathcal{D}$ and $op \in Op$.

For instance, a policy clause of the form:

$\leftarrow$ *User*(10), u.a, v.b, w.c, [u.a `child` v.b, u.a `right` w.c, v.b $\neq$ 10]

stipulates that a user with identity 10 is not allowed to get access to the information stored by attribute $a$ at node $u$, by attribute $b$ at node $v$, and by attribute $c$ at node $w$, all three together, if the scope constraint evaluates to true (cf. Example of Section 1). By the "all three together", it is intended that such a combined access – under *one or more queries* on the document – is disallowed to *User*(10).

*Definition (2):* At any node $u$ of $t$, if a set $\mathcal{D}_u(t)$ of defining functional relations is specified on the data stored at $u$ on $t$, then we also assume that $\mathcal{D}_u(t)$ is formulated as a set of Horn clauses of the form $u.a_{i_r} \leftarrow u.a_{i_1}, u.a_{i_2}, \ldots, u.a_{i_k}$, where the $a_j$ are attribute names at $u$.

The document $t$ being given, we shall drop references to $t$ as indices (or otherwise), in the definitions below.

*Definition (3):* For any query $Q =$`/`$C_1$`/`$C_2$`/`$\ldots$`/`$C_n$ that is being evaluated by a run of its associated transition graph $G_Q$ on the given document $t$, suppose a transition of the run, from a context node $u$ to a node $v$ on $t$, is a transition of a one-step transition graph $G_k^p$ for some $p, k$; then, by the *current query-step* of $Q$ at $u$, we mean the corresponding location step $step_k^p$ in $Q$ (notation of the previous section).

*Definition (4):* For any given query $Q$ launched by a given user with identity $id$, and a run of the associated graph $G_Q$, we define three sets of positive clauses – denoted as $Hist_{id}(v, Q)$, $Scope_{id}(v, Q)$ and $Access_{id}(v, Q)$ – at every node $v$ traversed by the run; these three sets are constructed, inductively in the following manner, wrt the the various queries successively launched by the user. Let $Q^{(0)}, Q^{(1)}, \ldots, Q^{(i)}, \ldots$ denote the sequence of the successive queries launched by the given user; the identity $id$ of the user once fixed, we shall omit it (for readability), in the constructions below (in these constructions, by '$root$', we mean the actual root node of the tree $t$):

**Step 0.** Set $i = 0$.

**Step 1.** Case $v$ is the root node of $t$:
$Scope(root, Q^{(i)}) = \emptyset$, by definition;
if $i = 0$, then $History(root, Q^{(i)})$ is a singleton set of the form $\{User(id) \leftarrow\}$ or $\{Cat(id) \leftarrow\}$;
if $i > 0$, $History(root, Q^{(i)})$ is defined as $Hist(root, Q^{(i-1)})$;
if $i = 0$, then $Access(root, Q^{(i)})$ is set to be empty;
if $i > 0$, $Access(root, Q^{(i)})$ is defined as $Hist(root, Q^{(i-1)})$.

**Step 2.** Case $v$ is a non-root node:
Let $u$ be the context node for the current transition of $G_{Q^{(i)}}$ to the node $v$. For all nodes $u'$ traversed prior to $u$ (including $u$), by the run of $G_{Q^{(i)}}$ evaluating $Q^{(i)}$ (i.e., the $i$-th query launched by the given user), assume having constructed the sets $Access(u', Q^{(i)})$, $Scope(u', Q^{(i)})$, and $Hist(u', Q^{(i)})$. The sets $Scope(v, Q^{(i)})$, $Access(v, Q^{(i)})$, and $Hist(v, Q^{(i)})$ are then constructed as follows:

1. Let `axis` be the axis of the current query-step at $u$, and suppose there is a node $u'$ on $t$ already traversed by the run such that $u'\,\texttt{axis}\,v$ holds on $t$; if $a$ (resp. $att$) is an attribute name at $u'$ (resp. at $v$), such that $u'.a\ \texttt{axis}\ \ v.att$ appears in the scope constraint of some policy clause, then create a positive 'scope clause':
   $[u'.a\ \texttt{axis}\ v.att] \leftarrow$;
   and define $Scope(v, Q^{(i)})$ as:
   $Scope(v, Q^{(i-1)}) \cup \{[u'.a\ \texttt{axis}\ v.att] \leftarrow\}$
2. For every $v.att$ appearing in the body of a policy clause in $\mathcal{P}$, such that $v.att$ is consistent with $\sigma\texttt{[}L\texttt{]}$ (where the current query-step is of the form $\texttt{axis::}\sigma\texttt{[}L\texttt{]}$), create a positive 'data-access clause' $v.att \leftarrow$, and define:
   $Access(v, Q^{(i)}) = Access(v, Q^{(i-1)}) \cup \{v.att \leftarrow\}$.
3. Define $Hist(v, Q^{(i)})$ as:
   $Hist(v, Q^{(i-1)}) \cup Scope(v, Q^{(i)}) \cup Access(v, Q^{(i)})$.

**Step 3.** Set $i = i + 1$, and GOTO **Step 1**.

*Definition (5):* In the presence of an access control policy $\mathcal{P}$ on $t$, a transition of the graph $G_Q$, from a context node $u$ to some node $v$ on $t$, assigns a selecting state to the node $v$ if and only if: $Hist(v, Q^{(i)}) \cup \mathcal{D}_v(t) \cup \mathcal{P} \not\models \perp$.

## 5 How does the method work?

Clausal resolution is essential for the step-wise evaluation technique for queries, proposed above, here, in addition to the usual first-order rules for resolving between positive and negative literals, we shall also employ some additional rules, such as those in the following non-exhaustive list (the Example below shows how they are used):

1. a literal $u.att$ can resolve with a literal (of the opposite sign) of the form $u.*$; the same also for literals (of opposite sign) of the form $User(id)$ and $User(*)$, etc.
2. a scope literal $[u.a\, \texttt{child}\, v.b]$ can resolve with a scope literal of the form $[v.b\, \texttt{parent}\, u.a]$, etc.
3. a negative scope literal $[u.a\ \texttt{axis}\ v.b]$ can resolve with a positive scope literal of the form $[u.a\ \texttt{axis}\ v.b\texttt{,}\ u.a \neq `val']$

**Example 2**. The history records at the nodes are necessary if the access control policy is not to be violated. For instance, consider the policy specified for the category *Adm*, in the Example of Section 1. Let us suppose, for instance, that the *grade* data stored at every *Course* node, is not a constant value (i.e., students from different grades follow the course). Suppose then that no history records are kept (for the data accessed), and that a given user of category *Adm* launches the following three queries, successively:
$Q_1 = \texttt{//}Student\texttt{/}@*$
$Q_2 = \texttt{//}Course\texttt{/}@grade$
$Q_3 = \texttt{//}Course\texttt{[}@grade \neq `g1'\texttt{]/parent::}Student\texttt{/}@name$
where $\texttt{//}*$ is short for $\texttt{/descendant::}*$ (as is standard in XPath). From the first query, one gets all the data stored at every *Student* node, in particular all the names of the students; from the second query, one retrieves all the *grade* data stored at every *Course* node; from the third, the user can get the names of all students, not registered for grade $g1$; (s)he can then deduce the name(s) of all students registered for grade $g1$, by complementation, and this will violate the given access policy, for the category *Adm*. The second query has only served to deduce that there *is* a *grade* attribute with value $g1$.

(Note: a query of the form:
$\texttt{//}Course\texttt{[}@grade = `g1'\texttt{]/parent::}Student\texttt{/}@name$,
or of the form: $\texttt{//}Student\texttt{[}@name = `n1'\texttt{]/}Course\texttt{/}@grade$, would have led to a violation of access control, without any need for history, and so would have produced no answer.)

Let us analyze the same three queries, when history records are kept. At the end of the evaluation of $Q_1$ and $Q_2$, these records will contain the following positive clauses:
$Adm(id) \leftarrow$,
$student.* \leftarrow$,
$course.grade \leftarrow$.
During the evaluation of the query $Q_3$, the following positive scope clause will get added to the history:

$$[course.grade \, \texttt{parent} \, student.name, course.grade \neq `g1'] \leftarrow .$$

The semantics of such an addition implies, in particular, that the scope constraint $[course.grade \neq `g1']$ evaluates to true. The set of all positive scope and data-access clauses thus generated, history inclusive, is inconsistent with the access policy specified for the category *Adm*; this is easily checked by resolution. As a result, the third query will *not* produce an answer, i.e., will not select any $student.name$.

One checks similarly, that the set of clauses $\mathcal{D}_v(t)$, modeling the defining functions for the data stored at any node $v$, are to be taken into account too (whenever specified).

**On the Completeness of the Approach** The step-wise evaluation method described above, based on the evaluating runs of the transition graph associated with the given query, is *deductively complete* in the sense of following proposition (the proof of which is straightforward, from definitions).

**Proposition 2.** *Let $\mathcal{P}$ be a set of negative clauses, implementing an access control policy on a given XML document $t$, for any given category of users. Then a piece of data, stored on $t$ as a value of an attribute* att *at a node with tagname $u$ on $t$, is accessible to a user of this category (under* some *XPath query)* if and only if *$u.att \leftarrow$ is element of a set $\mathcal{S}$ of positive data-access and scope clauses, that is consistent with $\mathcal{P}$.*

### 5.1 Expressive Power of the Clausal View

It is not hard to interpret the other approaches for access control policies, evoked in the Introduction. in terms of clauses. For that purpose, we adopt following terminology and notation. The terms *User* and *Category* will carry the same meanings as in Example 1 above. A *Key* $K$, on a document tree $t$, is defined as a (partial) function on the set $Nodes_t \cup Att$ composed of nodes on $t$ and attribute names, such that $K(x)$ is a non-empty subset of *Users*, when defined; thus $K(x)$ can be seen as a category, but has some additional properties, such as propagating access restrictions to children nodes (see below); we also assume given a set with two symbols: $Acc = \{-r, -R\}$, which will play the same access denial role as in [11]. The clauses below are the translations of the access policies in the aforementioned approaches; in these clauses $X$ stands for 'any' user in *User*, and $A, B$ stand for specified users; $u, v$ are nodes on any given document $t$; '$*$' stands for 'any' attribute at the nodes concerned; and when a key $K(x)$ is mentioned, it is assumed to be non-empty:

i) Clauses for local or recursive denial at a node:

- $\leftarrow X, u.* \, [Acc(u) = -r]$ /* local denial */
- $\leftarrow X, u.* \, [Acc(u) = -R]$ /* recursive denial */
- $\leftarrow X, v.* \, [Acc(u) = -R, \; u.* \; \texttt{child} \; v.*]$ /

ii) Clauses for key attribution to a node:

- $\leftarrow B, u.* \, [B \notin K(u)]$
- $\leftarrow B, v.* \, [B \notin K(u), \; u.* \; \texttt{child} \; v.*]$

iii) Clause for key attribution to an attribute at a node:
- $\leftarrow B, u.@att\,[B \notin K(att)]$

**Handling control key assignment rules** We made no explicit mention, in Section 4, on how to handle access control via key assignment rules. Here is how it can be done. We first observe that a query $/C_1/\ldots/C_n$ in canonical form has a non-empty answer on any $t$, only if the `axis` of its first elementary subquery $/C_1$ is either `child`, or `descendant`. This allows the history record for any user with some identity $B$, to account for a key $K$ assigned to a node $u$ on the document, by proceeding as follows, whatever be the query $Q$:
- if the access policy $\mathcal{P}$ contains a clause
$$\leftarrow B, u.*\,[B \notin K(u)],$$
then add the positive clauses $B \leftarrow,\;\; u.* \leftarrow$ to the set $History_B(u, Q)$, and do likewise at every $v$ below $u$ on $t$, whenever there is a transition of $G_Q$ from $u$ to $v$.

A policy that assigns the symbol $-r$ or $-R$, for local or recursive access denial at the nodes, is handled in the same manner.

## 6 Related Works, Conclusion

We have presented a simple logical framework for controlling the access to data stored at the nodes of XML (tree) format documents. The principal feature consists in modeling as a set of first-order clauses, any given access control policy, as well as the knowledge that a person can acquire, directly or indirectly, by querying the documents. The idea of constructing, for any given query $Q$, a node selecting transition graph $G_Q$ – such that the evaluation of $Q$ on any given document $t$ is done step-wise under a run of $G_Q$ on $t$ – has some similarity with the automaton associated with $Q$ in [7]; but unlike in that work, our graph $G_Q$ is not constructed in one single piece, and it serves a very different purpose too: that of controlling the access to data by controlling the run of $G_Q$ with a clausal resolution technique. [5] is a survey presenting the generalities on logical frameworks suited for access control, but the concern is not on query evaluation techniques. As for the work [3], access control via clauses is only evoked at the end, and the concern is not query evaluation either (but the study of a notion of non-interference of a given policy, based on type inference techniques).

We have shown that, in order to comply with the given access control policy, it is necessary to keep a history record for the data accessed by every given user (and also keep track of the functional relations specified between the attributes). We have shown that the complexity of our step-wise evaluation technique for queries is linear on the number of atomic location steps composing the given query, and the number of edges on the document, when no access control policies are given; in the presence of an access control policy, it seems possible to extend the linear evaluation strategy of Section 3.3, to get a complexity bound that is linear on the number of atomic location steps in $Q$, the number of edges of $t$, *and* the current size of the history record.

The work presented in this paper is still ongoing and has to pursued, in particular implemented and tested, on the practical side. On the theoretical side, it may be possible to consider the more general access control clauses of the form presented in [1], and still preserve our step-wise evaluation technique for queries. We also hope to address an altogether different issue: covering the case where the document is represented as a DAG, and not as a tree (e.g., as when IDs and IDREFs are part of the specification); the transition graph $G_Q$ may then have to be constructed differently (following some of the ideas we developed in [4]).

## References


1. M. Abadi, *Logic in Access Control* In Proc. LICS'03, IEEE, pp. 228–233,
2. M. Abadi, B. Warinschi, *Security Analysis of Cryptographically Controlled Access to XML Documents* In Proc. of Int. Conf. PODS'05, pp. 108–117, ACM, June 2005,
3. V. Benzaken, M. Burelle, G. Castagna, *Information Flow Security for XML Transformations*, In Proc. of the 8th Int. Conf. ASIAN'03, pp. 33–53, LNCS 4246, Springer-Verlag, 2003,
4. B. Fila, S. Anantharaman, *Automata for Positive Core XPath Queries on Compressed Documents*, In Proc. of the 13th Int. Conf LPAR'06, pp. 467–481, LNAI 4246, Springer-Verlag, 2006,
5. I. Fundulaki, M. Marx, *Specifying Access Control Policies for XML Documents with XPath* In Proc. of SACMAT'04, pp. 61–69, ACM, 2004.
6. 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.
7. T. Green, G. Miklau, M. Onizuka, D. Suciu, *Processing XML Streams with Deterministic Automata*, In ACM Transactions on Database Systems, 29(4):752–788, 2004.
8. M. Marx, *XPath and Modal Logics for Finite DAGs*. In Proc. of TABLEAUX'03, pp. 150–164, LNAI 2796, 2003.
9. W. Martens, F. Neven, *On the complexity of typechecking top-down XML transformations*, In Theoretical Computer Sc., 336(1): 153–180, 2005.
10. G. Miklau, D. Suciu, *Controlling access to published data using cryptography*, In Proc. 29th Int. Conf. VLDB'03, pp. 898–909, 2003.
11. 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.
12. World Wide Web Consortium, *XML Path Language (XPath Recommendation)*, Available at: `http://www.w3c.org/TR/xpath/`