Updates and Views Dependencies in Semi-structured Databases

Françoise GIRE
CRI Paris 1 Panthéon Sorbonne University
90 rue de Tolbiac, 75013
Paris, France
gire@univ-paris1.fr

Hicham IDABAL
CRI Paris 1 Panthéon Sorbonne University
90 rue de Tolbiac, 75013
Paris, France
hidabal@univ-paris1.fr

ABSTRACT

In this paper we study the classical problem of the impact of an update on a view defined over semi-structured data. We adopt the following working hypotheses: (i) the source document is modelized by an unranked, labeled, ordered tree, (ii) a view $\mathcal{V}$ is a tree query whose evaluation on the source document provides a desired partial view of the document, (iii) a class of updates $\mathcal{C}$ is also given by a tree query selecting the nodes to modify. We then study the following problem: given a view query $\mathcal{V}$ and a class of updates $\mathcal{C}$, is it possible to detect if the view $\mathcal{V}$ is independent of each update q in $\mathcal{C}$? We show that the problem is in general PSPACE-hard. We propose a sufficient condition evaluable in polynomial time ensuring the independence of a view $\mathcal{V}$ with respect to a class of updates $\mathcal{C}$. We then consider the class of linear view queries for which the problem becomes polynomial.

Keywords Semi-structured Data, XML queries, Updates, Views, Tree Automata

1. INTRODUCTION

Work relating to the management of semi-structured data is increasingly growing in the literature, motivated by the role of XML as a standard to exchange information on the Web. This topic constitutes today one of the most active fields of research in data processing. Static aspects of semi-structured data management are largely present in the literature, but dynamic aspects relating to the evolution over time of semi-structured data have just started being studied. In the context of XML on the Web, the constantly changing nature of the data has to be taken into account. In this paper we investigate the problem of analyzing the impact of an update operation on a view defined on semi-structured data. For personalization or confidentiality needs, a view gives a partial presentation of the information contained in one or more source documents. Thus a view is modelized by a query whose evaluation on the data sources provides the desired partial presentation. In case of wide source documents, the cost of the view evaluation may be high. Therefore avoiding, when possible, a view re-computation, is of great interest for query optimization. The problem of analyzing the impact of updates on views precisely consists in detecting if the result of the view query evaluation has been modified after an update on the source documents.

The addressed problem is classical for relational databases [10, 12, 21], it has been also examined for object databases [2, 20] and for semi-structured data in several works [1, 14, 16, 17, 19, 23]. However in a majority of these works, the problem has been studied by considering that the source documents on which the view is evaluated are available and the implemented methods are using these source documents. We adopt here a different approach to analyze this problem: we suppose that the source documents aren't a priori available and that only the view query $\mathcal{V}$ and the update operation are given. We also modelize the update by a query $\mathcal{C}$, selecting nodes to be modified. Insofar as we don't specify how the nodes selected by $\mathcal{C}$ will be modified, $\mathcal{C}$ represents a class of updates, namely all updates modifying exactly the nodes selected by $\mathcal{C}$. The statement of the problem we are studying is then the following: given a view $\mathcal{V}$ and a class $\mathcal{C}$ of updates, is it possible to determine if the view $\mathcal{V}$ is independent with respect to $\mathcal{C}$, i.e. if each update q in $\mathcal{C}$ does not have any impact on the evaluation of $\mathcal{V}$, on every source document $\mathcal{T}$. If a schema $\mathcal{S}c$ constraining the source documents is known, $\mathcal{S}c$ will hopefully be used to refine the impact analysis and to detect more cases of independence. We thus also study the following version of the preceding problem: given a view $\mathcal{V}$, a class of updates $\mathcal{C}$ and a schema $\mathcal{S}c$, is it possible to determine if the view $\mathcal{V}$ is independent with respect to $\mathcal{C}$ in the context of $\mathcal{S}c$, i.e. if each update q in $\mathcal{C}$ does not have any impact on the evaluation of $\mathcal{V}$, on every source document $\mathcal{T}$ valid with respect to $\mathcal{S}c$.

Our main result is to provide a sufficient condition for assuring the independence between a view $\mathcal{V}$ and a class of updates $\mathcal{C}$ in the context of a schema $\mathcal{S}c$. This condition is evaluable in polynomial time in the sizes of $\mathcal{V}$, $\mathcal{C}$ and $\mathcal{S}c$. We also show that, in general, the problem is PSPACE-hard. Finally we consider the class of linear view queries for which the problem becomes polynomial (without any restriction on the class $\mathcal{C}$ of updates).

Related work

Some other works are similar to ours but take place in the different context of optimization of query processing: in [4] the authors exploit materialized XPath views in order to expedite processing of XML queries and they develop an XPath matching algorithm to determine when such views can be used to answer an user query; in [13] maximal contained rewriting under constraint is considered; also in [3], a sound and complete view-based rewriting algorithm for nested XQuery queries is proposed in the presence of structural and integrity constraints, which produces an algebraic plan combining tree pattern views. These works differ from ours on several aspects: (i) although similar, the rewriting query problem using materialized views is different from the independence query problem (ii) the chosen query language is mostly XPath or XQuery while we opt here for the more conceptual query language based on regular tree queries; regular tree queries can express some fragments of XPath queries but do not express XPath queries containing disjunctive conditions; however thanks to its regular pattern matching feature, a regular tree query can express queries not expressible in XPath (iii) some additional stored information issued from the source documents, is used in the proposed algorithms while we don't use any kind of such information but only specifications of the view query, of the update operations and of the schema when available.

In the context of updates, other works have to be mentioned. In [5] the authors are interested in determining statically whether updates generated by a program can be applied before all querying is completed and they provide an algorithm testing sufficient conditions for this property. Although their analysis uses quite different techniques than ours, involving satisfiability of particular systems of equations, the problem studied in [5] is closely related to our independence query problem: actually, changing the update evaluation ordering does not violate the semantics of the program, if these updates do not interfere with each other. Similarly to our work, [18] studies the problem of detecting conflicts among XML update operations specified by tree patterns of $P^{*,//,[]}$ introduced in [15]. The problem is proved as NP-complete and a polynomial algorithm is given when the read pattern is linear. In [7] the same problem is addressed with the assumption that input documents are typed by an XML schema. Again, our approach mainly differs from all these works by the choice of the regular tree query model as conceptual query language which impacts on the obtained results and the involved resolution techniques.

Outline The rest of the paper is organized as follows. Section 2 presents the notions of independence between views and update classes. Section 3 defines the regular tree query model used to modelize views and update classes. In section 4 we analyze the independence query problem, and we give in section 5 a sufficient condition for the independence. We conclude in section 6.

2. PRELIMINARIES

In this work semi-structured data are XML documents which are modelized by unranked, labeled, ordered trees (Figure 1). In this model, character data in XML elements are ignored, only elements defining the document structure are represented as nodes of the unranked labeled tree. Formally, a document $\mathcal{T}$ is a pair $\mathcal{T}=(D,\lambda)$ where D is a tree domain, (i.e. D is a subset of $\mathbb{N}^*$ containing the empty word and satisfying $\forall i \in \mathbb{N}$, $wi \in D \Rightarrow (w \in D$ and $wj \in D, \forall j < i)$) and λ associates a label $a \in \Sigma$ with each node w in D. In the following, the domain D of $\mathcal{T}=(D,\lambda)$ will be denoted by $\mathcal{N}(\mathcal{T})$. For each node w in $\mathcal{N}(\mathcal{T})$, we denote by $\mathcal{T}(w)$, the sub-tree rooted at w in $\mathcal{T}$ and defined by $\mathcal{T}(w) = (D_w, \lambda_w)$ with $D_w = \{wv/v \in \mathbb{N}^*\} \cap D$ and λ_w is the restriction of λ on D_w. For technical reasons, we adopt the convention that the root is labeled with the symbol '/' in every document $\mathcal{T}$.

2.1 Views and Updates

Views A view can be defined as a partial presentation and reorganization of some source data, for personalization or confidentiality needs. In the case of XML documents, the execution of a view consists in two main steps: the first step extracts from source documents a set of relevant nodes containing the desired information; the second step reorganizes the result obtained at the first step in order to meet the required personalized structure. Thus we can model a view $\mathcal{V}$ as a composition of two applications : $\mathcal{V} = h \circ t$, where application t selects the nodes to be extracted from the source documents and application h proceeds to the reorganization of these nodes to obtain the final result. This application h doesn't play any role in the independence analysis between a view and an update, so we identify a view with its first node extraction step i.e. $\mathcal{V} \simeq t$. More precisely, we consider in this work that (1) the source documents are consisting in a single XML document $\mathcal{T}$ and that (2) a view $\mathcal{V}$ is a n-ary query expressing the conditions, for a tuple of $\mathcal{T}$'s nodes, to be extracted by $\mathcal{V}$. We consider that the result of the extraction of a node i returns the sub-tree $\mathcal{T}(i)$, rooted at i in $\mathcal{T}$. So the execution of a n-ary view query $\mathcal{V}$ on the document $\mathcal{T}$ returns a set of sub-tree tuples $(\mathcal{T}(i_1), ..., \mathcal{T}(i_n))$ corresponding to the node tuples $(i_1, ..., i_n)$ selected by $\mathcal{V}$.

Example 1: Let us consider the document $\mathcal{T}$ of Figure 1 and the following binary view query $\mathcal{V}$: "Give the pairs (Title, Author) for articles published in a journal". Its evaluation is $\{(t_1, s_1), (t_1, s_2)\}$ with $t_1 = \mathcal{T}(00110)$, $s_1 = \mathcal{T}(00111)$ and $s_2 = \mathcal{T}(00112)$, i.e. the two pairs of sub-trees built from the title and the two authors of the article corresponding to node 0011.

Updates In this work we assume that an update on a XML document $\mathcal{T}$ consists in (1) selecting a set of nodes in $\mathcal{T}$ to be updated and (2) replacing the sub-tree $\mathcal{T}(w)$ rooted at each selected node w by a new sub-tree. This modelization covers most of current update operations, including inserting/deleting sub-tree operations: actually such operations can be viewed as updating the father nodes of the insertion/deletion positions. Hence we define an update q of a semi-structured document as the composition $q = f \circ \mathcal{C}$ of two applications f and $\mathcal{C}$: application $\mathcal{C}$ selects the set of nodes w to be updated and application f performs the updates by replacement of the sub-trees $\mathcal{T}(w)$ rooted at these nodes w. Application $\mathcal{C}$ represents in fact a class of updates: two updates belong to the same class $\mathcal{C}$ if and only if they are defined with the same node selecting application $\mathcal{C}$.

Example 2: Let us consider the two following update queries q_1:"Modify the authors of an article by adding a phone number" and q_2:"Modify the authors of an article by changing element Name into two elements (Last Name, First Name)". Queries $q1$ and $q2$ belong to the same update class $\mathcal{C}$ selecting the authors of an article for a modification.

2.2 Independence between views and update classes

Impact We say that an update q has an impact on a view $\mathcal{V}$ of a document $\mathcal{T}$ if and only if the evaluations of $\mathcal{V}$ on $\mathcal{T}$ before and after the update q do not produce the same result, i.e $\mathcal{V}(q(\mathcal{T})) \neq \mathcal{V}(\mathcal{T})$. Thus the update queries q_1 and q_2 of Example 2 clearly have an impact on the view query $\mathcal{V}$ of Example 1, since they modify the extracted sub-trees s_1 and s_2.

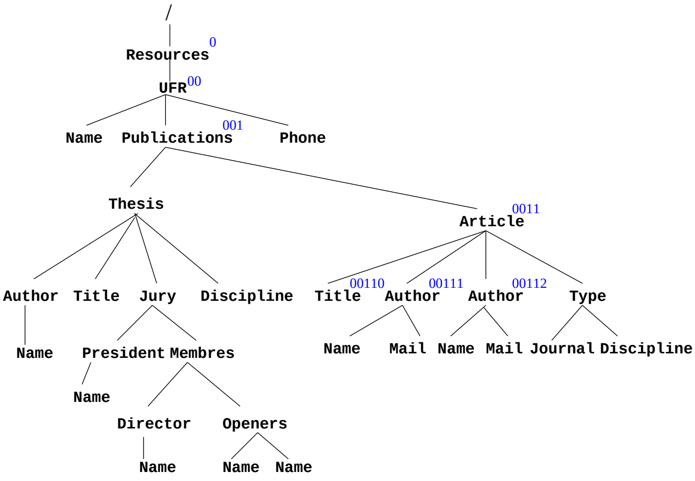


Figure 1: A semi-structured document $\mathcal{T}$

Independence The goal of this work is to directly analyze both the view query $\mathcal{V}$ and the update query q in order to detect an eventual impact of q on the result of $\mathcal{V}$'s evaluation. It is important to note that our objective is to detect this impact independently of the source document which is supposed not available during the analysis. In addition, in order to simplify the problem, we suppose that the specification of the function f, performing the update, is unknown and therefore that f can be of any type. Hence we adopt to focus us on the detection of impacts of a whole class $\mathcal{C}$ of updates (rather than of a particular update q) with respect to a view evaluation. The problem is thus formally stated as follows: $\mathcal{V}$ is independent with respect to the update class $\mathcal{C}$ if and only if for every source document $\mathcal{T}$, for every update q in $\mathcal{C}$, q doesn't impact the view $\mathcal{V}$ of $\mathcal{T}$.

Example 3 (impact/no impact) : Let us consider the following view query $\mathcal{V}'$ on the document $\mathcal{T}$ (Figure 1): "Give the pairs (Title, Author Name) for articles published in a journal". This query differs from the query $\mathcal{V}$ of Example 1 because only the Name node of an author, instead of the whole Author node, is extracted with the Title node of an article. Update q_2 (Example 2) has clearly an impact on this view since it modifies author names. Hence the class $\mathcal{C}$, selecting the article authors for modification, has an impact on the view $\mathcal{V}'$ since there exists an update (q_2) in this class that impacts the result of $\mathcal{V}'$. If we consider the class $\mathcal{C}'$ which selects the thesis authors (instead of the article authors) for modification, view $\mathcal{V}'$ is clearly independent of the class $\mathcal{C}'$.

Independence in the context of a schema In many cases, a schema constraining the source documents structure is available. This additional information can then improve the independence analysis. Let us denote by $\mathtt{valid}(\mathcal{S}c)$ the set of valid documents with respect to a schema $\mathcal{S}c$. In this context the independence definition is modified as follows: $\mathcal{V}$ is independent with respect to an update class $\mathcal{C}$ in the context of a schema $\mathcal{S}c$ if and only if : $\forall\ \mathcal{T} \in \mathtt{valid}(\mathcal{S}c)$, $\forall$ $\mathbf{q} \in \mathcal{C}$ with $\mathbf{q}(\mathcal{T}) \in \mathtt{valid}(\mathcal{S}c)$, $\mathbf{q}$ doesn't impact the view $\mathcal{V}$ of $\mathcal{T}$.

3. TREE QUERIES

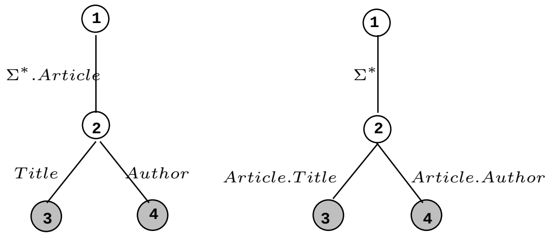
The preceding definitions of views and updates share a common selection process of nodes (or of tuples of nodes) that plays a main role. In this work we choose to use the concept of tree query to model this process. Let us introduce intuitively this concept by considering the binary query "Give the pairs (Title, Author) of Article nodes", and its evaluation on the semi-structured document $\mathcal{T}$ of Figure 1. This query can be represented by the tree query $\mathscr{R}$ shown in Figure 3 which specifies the conditions, for a pair (i, j) of $\mathcal{T}$'s nodes, to be extracted: (i, j) is extracted from $\mathcal{T}$ if it is possible to find a mapping p of the tree $\mathscr{R}$ on the document $\mathcal{T}$, such that (see Figure 3) (1) grey nodes 3 and 4 of $\mathscr{R}$ are precisely associated by p with nodes i and j i.e. $p(3) = i$, $p(4) = j$ and, (2) for each edge (i_1, i_2) of $\mathscr{R}$, there exists a path in the source document $\mathcal{T}$ between $p(i_1)$ and $p(i_2)$ whose sequence of labels satisfies the constraints expressed by the regular expression labeling the edge (i_1, i_2) in $\mathscr{R}$.

Formally, a tree query on the alphabet Σ is a tree whose edges are labeled by regular expressions on Σ. More precisely:

Definition 1. *Let Σ be a finite alphabet of labels. A n-ary tree query over Σ is defined by : $\mathscr{R} = (\Sigma, N, M, I, \mathcal{E})$ where:*
- (N, M) is a tree with N as finite set of nodes and $M \subseteq N \times N$ as set of edges.
- $\mathcal{E} : M \longrightarrow REG(\Sigma)$ is an application which associates with each edge (i, j) of M a regular expression of $REG(\Sigma)$ denoted by $\mathcal{E}_{(i,j)}$
- I is a tuple of specific nodes $(i_1, ..., i_n)$ representing the selected nodes.

In the later, we denote by $Out(i)$ the set of edges outgoing from node i in $\mathscr{R}$ and we only consider tree queries $\mathscr{R}$ in which $Out(1) = 1$. The size of $\mathscr{R}$ denoted by $|\mathscr{R}|$ is defined by: $|\mathscr{R}| = |N| + \Sigma_{e \in M} |\mathcal{A}_e|$ where $\mathcal{A}_e$ is a word automaton associated to the regular expression $\mathcal{E}_e$ and $|\mathcal{A}_e|$ denotes the size of $\mathcal{A}_e$.

Figure 2 gives two examples of a tree query.



(a) *Tree query $\mathscr{R}$* **(b)** *Tree query $\mathscr{R}'$*

Figure 2: Tree queries

In the case of update operations, we restrict our approach to unary tree queries whose updated node is a leaf. As we will see later this assumption allows us to find a polynomial sufficient condition assuring the independence of a view with respect to a class of updates.

3.1 Evaluation of Tree queries

The evaluation of a n-ary tree query on a semi-structured document uses the concept of mapping.

Definition 2. *A mapping of a tree query $\mathscr{R} = (\Sigma, N, M, I, \mathcal{L})$ on a semi-structured document $\mathcal{T}$ is an application p from N to $\mathcal{N}(\mathcal{T})$ such that:*

- *If r is the root node of $\mathscr{R}$ then $p(r)$ is the root node of $\mathcal{T}$ labeled with '/'*
- *$\forall\ n \in N$, there exists $m \in N(\mathcal{T})\ /\ p(n) = m$*
- *$\forall i, j \in N$, if $i \leq j$ then $p(i) \leq p(j)$*
- *If $p(i) = p(j)$ then $i = j$*
- *$\forall e = (i, j) \in M$, there exists in $\mathcal{T}$ a path p_{ij} from $p(i)$ to $p(j)$, excluding $p(i)$ and including $p(j)$, such that: (a) the sequence denoted by $p(e)$ of the labels occurring on this path is a word of the language defined by $\mathcal{L}_{(i,j)}$, and (b) if $e_1 = (i, j)$ and $e_2 = (i, k)$ are two distinct outgoing edges from node i then the paths p_{ij} and p_{ik} have no common prefix.*

Let us remark that the images, in a semi-structured document, of mappings of the tree queries $\mathscr{R}$ and $\mathscr{R}'$ of Figure 2 are quite different: the first ones extract pairs of nodes (Title, Author) that are children of a same Article node, while the second ones extract pairs of nodes (Title, Author) that are children of two distinct Article nodes.

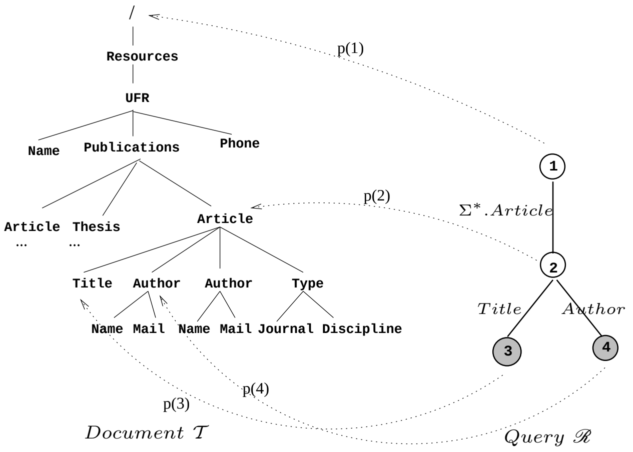


Figure 3: Mapping of query $\mathscr{R}$ on document $\mathcal{T}$

Trace of a tree query with respect to a mapping The trace of $\mathscr{R}$ in $\mathcal{T}$ with respect to the mapping p is the smallest sub-tree of $\mathcal{T}$ containing the image $p(N)$. We denote it by $\texttt{trace}_p(\mathscr{R}, \mathcal{T})$.

Evaluation of a tree query Let $\mathscr{R}$ be a tree query , $\mathcal{T}$ a semi-structured document and $\mathcal{P}$ the set of all mappings of $\mathscr{R}$ on $\mathcal{T}$.

- The result, denoted by $\mathscr{R}_p(\mathcal{T})$, of the evaluation of $\mathscr{R}$ on $\mathcal{T}$ with respect to the mapping p of $\mathcal{P}$, is the tree tuple $\mathscr{R}_p(\mathcal{T}) = (\mathcal{T}(p(i_1)), \ldots, \mathcal{T}(p(i_n)))$ where $I = (i_1, ..., i_n)$ is the tuple of selected nodes of $\mathscr{R}$.
- The result, denoted by $\mathscr{R}(\mathcal{T})$, of the evaluation of $\mathscr{R}$ on $\mathcal{T}$, is $\mathscr{R}(\mathcal{T}) = \bigcup_{p \in \mathcal{P}} \mathscr{R}_p(\mathcal{T})$

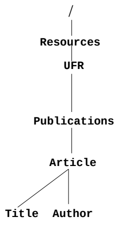


Figure 4: Trace of the tree query $\mathscr{R}$

4. THE INDEPENDENCE PROBLEM

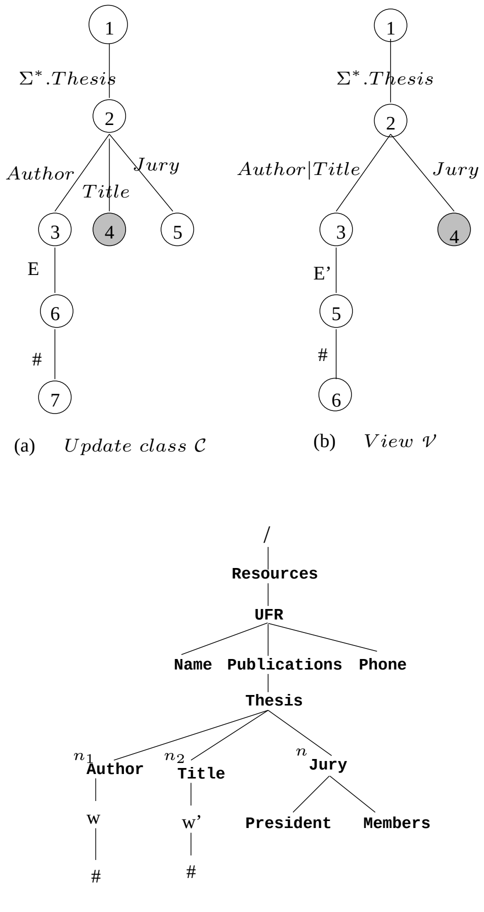
4.1 PSPACE-hardness

Proposition 1. *Deciding whether a view $\mathcal{V}$ is independent with respect to an update class $\mathcal{C}$ is a PSPACE-hard problem .*

Proof. We reduce the well-known PSPACE-hard problem of the inclusion of two regular expressions, into the problem of independence. Let us consider two regular expressions E and E'. We define two tree queries $\mathcal{V}$ and $\mathcal{C}$ as shown in Figure 5 where '#' is a new label occurring neither in E nor in E'. We prove that $\mathcal{V}$ is dependent on $\mathcal{C}$ iff $E \nsubseteq E'$.

Suppose that $\mathcal{V}$ is dependent on $\mathcal{C}$. There exists a document $\mathcal{T}$ and an update $q \in \mathcal{C}$ such that $\mathcal{V}(q(\mathcal{T})) \neq \mathcal{V}(\mathcal{T})$. Therefore either (a) a node n of $\mathcal{T}$ labeled by 'Jury' is extracted by $\mathcal{V}$ i.e. $n \in \mathcal{V}(\mathcal{T})$ but is no more extracted after the update q i.e. $n \notin \mathcal{V}(q(\mathcal{T}))$ or (b) a new node n labeled by 'Jury' is extracted by $\mathcal{V}$ after the update q i.e. $n \in \mathcal{V}(q(\mathcal{T}))$ whereas it wasn't before i.e. $n \notin \mathcal{V}(\mathcal{T})$.
In case (a), the fact $n \notin \mathcal{V}(q(\mathcal{T}))$ implies that n belongs to the image of some mapping p of the tree query $\mathcal{C}$ on $\mathcal{T}$: therefore n has got sibling nodes n_1 and n_2, respectively labeled by 'Author' and 'Title', that are the images by p of the nodes 3 and 4 of $\mathcal{C}$. Clearly, if $E \subseteq E'$, then the subtree rooted at n_1 in $\mathcal{T}$ after the update q still fulfills the conditions required by the tree query $\mathcal{V}$ for the extraction of n, contradicting the fact $n \notin \mathcal{V}(q(\mathcal{T}))$.
Similarly in case (b), the facts $n \in \mathcal{V}(q(\mathcal{T}))$ and $n \notin \mathcal{V}(\mathcal{T})$ imply that n belongs to the image of some mapping p of the tree query $\mathcal{C}$ into $\mathcal{T}$: therefore n has got sibling nodes n_1 and n_2, respectively labeled by 'Author' and 'Title', that are the images by p of the nodes 3 and 4 of $\mathcal{C}$. Again if $E \subseteq E'$, then the subtree rooted at n_1 in $\mathcal{T}$ before the update q fulfills the conditions required by the tree query $\mathcal{V}$ for the extraction of n, contradicting the fact $n \notin \mathcal{V}(\mathcal{T})$.

Conversely if $E \nsubseteq E'$, there exists a word w in $L(E)$ that doesn't belong to $L(E')$. Let w' be a word of $L(E')$. We consider the tree $\mathcal{T}_0$ shown in Figure 5 where: n_1, n_2 and n denote the nodes respectively labeled by 'Author', 'Title' and 'Jury', and w (respectively w') denotes the sequence of labels encountered on the path from the Author (respectively Title) node to the # node. Because w' belongs to $L(E')$, n is extracted by $\mathcal{V}$ before any update q of $\mathcal{C}$. Because w belongs to $L(E)$, n_2 is updated by any update of $\mathcal{C}$. Let us now consider the update q of $\mathcal{C}$ that removes the path $w'\#$ from the subtree rooted at n_2. Then n is no more extracted by $\mathcal{V}$ after q because w doesn't belong to $L(E')$. So $\mathcal{V}$ is dependent on $\mathcal{C}$.



(a) *Update class $\mathcal{C}$*

(b) *View $\mathcal{V}$*

(c) *Tree $\mathcal{T}_0$*

Figure 5:

4.2 An independence criterion

According to the definitions of the preceding sections, a view query $\mathcal{V}$ is dependent on an update class $\mathcal{C}$ in the context of a schema $\mathcal{S}c$ if and only if there is $\mathcal{T} \in valid(\mathcal{S}c)$, there is $q \in \mathcal{C}$ with $q(\mathcal{T}) \in valid(\mathcal{S}c)$ and there is a tuple $\overrightarrow{n}$ of sub-trees in $\mathcal{T}$ satisfying: (1) $\overrightarrow{n}\in \mathcal{V}(\mathcal{T})$ *and* $\overrightarrow{n}\notin \mathcal{V}(q(\mathcal{T}))$ or (2) $\overrightarrow{n}\notin \mathcal{V}(\mathcal{T})$ *and* $\overrightarrow{n}\in \mathcal{V}(q(\mathcal{T}))$.

Equation (1) means that the extracted tuple $\overrightarrow{n}$ disappears after the update operation, thus there exists a trace $trace_p(\mathcal{V}, \mathcal{T})$ of $\mathcal{V}$ in $\mathcal{T}$ with respect to some mapping p satisfying one of the following conditions:

- A node of $trace_p(\mathcal{V}, \mathcal{T})$ has been modified by q and the tuple $\overrightarrow{n}$ is no more extracted after the update (case 1 of Figure 6).

- A subtree of $\overrightarrow{n}$ was modified by q and $\overrightarrow{n}$ does not appear any more in the result of $\mathcal{V}$'s evaluation (case 2 of Figure 6).

Equation (2) means that a new tuple $\overrightarrow{n}$ is extracted after the update. This happens because the update q has generated a new trace $trace_p(\mathcal{V}, q(\mathcal{T}))$ of $\mathcal{V}$ in $q(\mathcal{T})$ allowing the extraction of $\overrightarrow{n}$ (case 3 of Figure 6).

Notation : If $\overrightarrow{n}= (\mathcal{T}_1, ..., \mathcal{T}_n)$ is a tuple of sub-trees, we denote $\mathcal{N}(\overrightarrow{n}) = \cup_{i=1}^{n}\mathcal{N}(\mathcal{T}_i)$.

DEFINITION 3. *Let $\mathcal{L}$ be the set of trees $\mathcal{T}$ such that:*
(i) $\mathcal{T} \in \mathtt{valid}(\mathcal{S}c)$
(ii) There is a trace $trace_p(\mathcal{V}, \mathcal{T})$ of $\mathcal{V}$ in $\mathcal{T}$ with respect to some mapping p of $\mathcal{V}$ on $\mathcal{T}$,
(iii) There is a trace $trace_{p'}(\mathcal{C}, \mathcal{T})$ of $\mathcal{C}$ in $\mathcal{T}$ with respect to some mapping p' of $\mathcal{C}$ on $\mathcal{T}$,
(iv) $p'(I_c) \in \mathcal{N}(trace_p(\mathcal{V}, \mathcal{T})) \cup \mathcal{N}(\mathcal{V}_p(\mathcal{T}))$

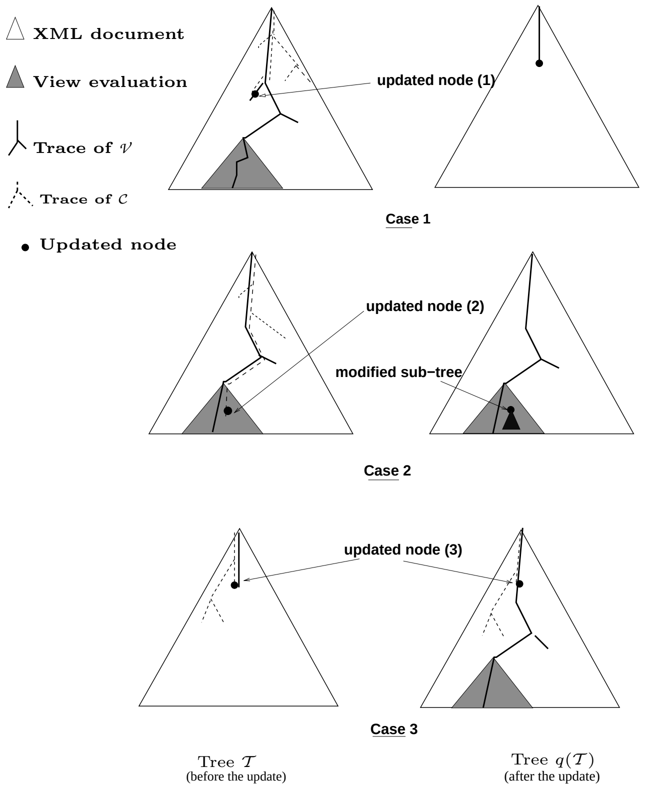


Figure 6:

PROPOSITION 2. *If $\mathcal{L} = \varnothing$ then $\mathcal{V}$ is independent with respect to $\mathcal{C}$ in the context of $\mathcal{S}c$.*

PROOF. Suppose that $\mathcal{V}$ is dependent on $\mathcal{C}$ in the context of $\mathcal{S}c$, then equation (1) or equation (2) holds. The first one implies the existence of a mapping p' of $\mathcal{C}$ on $\mathcal{T}$ and a mapping p of $\mathcal{V}$ on $\mathcal{T}$ such that $p'(I_c) \in \mathcal{N}(trace_p(\mathcal{V}, \mathcal{T})) \cup \mathcal{N}(\mathcal{V}_p(\mathcal{T}))$. So $\mathcal{L} \neq \varnothing$.
The second one implies the existence of a mapping p' of $\mathcal{C}$ on $\mathcal{T}$ and a mapping p of $\mathcal{V}$ on $q(\mathcal{T})$. Since the updated node is a leaf in the tree query $\mathcal{C}$, the trace of $\mathcal{C}$ in $\mathcal{T}$ remains in $q(\mathcal{T})$. So there exists a mapping p'' of $\mathcal{C}$ on $q(\mathcal{T})$ (identical to p') with $p''(I_c) \in \mathcal{N}(trace_p(\mathcal{V}, q(\mathcal{T})))$. Therefore $q(\mathcal{T}) \in \mathcal{L}$ and $\mathcal{L} \neq \varnothing$.
Remark The criterion $\mathcal{L} = \varnothing$ is not required for the independence of a view with respect to an update class as shown by the tree queries $\mathcal{V}$ and $\mathcal{C}$ of Figure 5 when E, E', w and w' are empty words: the tree $\mathcal{T}_0$ belongs to $\mathcal{L}$ (and therefore $\mathcal{L} \neq \varnothing$) because it satisfies the conditions of Definition 3; however it is easy to verify that $\mathcal{V}$ is independent with respect to $\mathcal{C}$.

4.3 Linear view queries

In the particular case where the view query is defined by a linear tree, the independence criterion exhibited in the previous section becomes a necessary and sufficient condition for the independence.

Definition 4. *A linear query is a query defined by a linear tree.*

Proposition 3. *Let $\mathcal{V}$ be a linear view query, $\mathcal{C}$ an update class, and $\mathcal{S}c$ a schema. $\mathcal{V}$ is independent with respect to $\mathcal{C}$ in the context of $\mathcal{S}c$ iff $\mathcal{L}$ is empty.*

Proof. By Proposition 2, we only have to prove that the criterion $\mathcal{L} = \emptyset$ is a necessary condition for the independence. Suppose that $\mathcal{L} \neq \varnothing$ and let $\mathcal{T}$ be a tree in $\mathcal{L}$. We show that $\mathcal{T}$ is a witness of the dependence of $\mathcal{V}$ with respect to $\mathcal{C}$: according to the definition of $\mathcal{L}$, there are on $\mathcal{T}$, two mappings p and p' of $\mathcal{V}$ and $\mathcal{C}$ respectively, and an updated node $n = p'(I_c)$ that belongs either to $\mathcal{N}(trace_p(\mathcal{V}, \mathcal{T}))$ (case (a) of Figure 7) or to $\mathcal{N}(\mathcal{V}_p(\mathcal{T}))$ (case (b) of Figure 7); because $\mathcal{V}$ is linear, in both cases, n is an ancestor or a descendant of the extracted node m. So in both cases, we specify the update q of $\mathcal{C}$ as shown in Figure 7: q keeps all the structure of the subtree rooted at n and only adds, as a common descendant of n and m, a new leaf node ν that wasn't occurring before in the subtree extracted by $\mathcal{V}$. We clearly have: $\mathcal{V}(q(\mathcal{T})) \neq \mathcal{V}(\mathcal{T})$. Therefore $\mathcal{V}$ is dependent on $\mathcal{C}$.

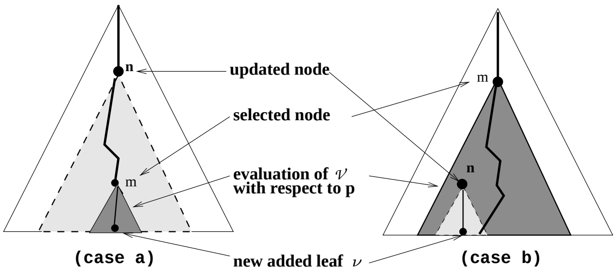


Figure 7: The linear view case

5. CHECKING THE INDEPENDENCE CRITERION

To check the vacuity of $\mathcal{L}$, we first define a tree automaton $\mathcal{A}$ recognizing $\mathcal{L}$. As we will see later, the size of this automaton $\mathcal{A}$ can be proved as polynomial in the sizes of $\mathcal{V}$, $\mathcal{C}$ and $\mathcal{S}c$. Because the regularity of $\mathcal{L}$, $\mathcal{L}$'s vacuity test is feasible in polynomial time with respect to the size of an automaton recognizing $\mathcal{L}$. This allows us to deduce that the independence criterion, $\mathcal{L} = \emptyset$, is testable in polynomial time with respect to the sizes of $\mathcal{V}$, $\mathcal{C}$ and $\mathcal{S}c$.

5.1 The trace automaton

Given a tree query $\mathscr{R} = (\Sigma, N, M, I, \mathscr{E})$, we first define an automaton $\mathcal{A}_{\mathscr{R}} = (\Sigma, Q, \delta, F)$ (where Q is the set of states, δ the transition function and F the set of final states) that recognizes the set of trees containing a trace of $\mathscr{R}$. The construction of $\mathcal{A}_{\mathscr{R}}$ uses finite word automata associated with the regular expressions occurring in $\mathscr{R}$: for each regular expression $\mathscr{E}_e$ (where e is an edge in $\mathscr{R}$), let us denote by $L(\mathscr{E}_e)$ the rational language defined by $\mathscr{E}_e$; we use in fact the rational languages $\widetilde{L(\mathscr{E}_e)} = \{\ \tilde{w}\ /\ w \in L(\mathscr{E}_e)\}$, each of them consisting of the set of mirror images of words of $L(\mathscr{E}_e)$. Intuitively, given a tree $\mathcal{T}$ and a mapping p of $\mathscr{R}$ in $\mathcal{T}$, a run of $\mathcal{A}_{\mathscr{R}}$ on $\mathcal{T}$ will simulate (see Figure 8) the runs of word automata recognizing the words $\widetilde{p(e)} = \sigma_1\sigma_2...\sigma_{n-1}$.

We consider, for each edge e, a finite word automaton $\mathcal{A}_e = (\Sigma, Q_e, \delta_e, t_e^0, f_e)$ recognizing $\widetilde{L(\mathscr{E}_e)}$. Without loss of generality, we assume the three following properties:
(i) the sets $\{Q_e, e \in M\}$ are pairwise disjoined,
(ii) for each e, $\mathcal{A}_e$ has an unique initial state t_e^0 and an unique final state f_e and,
(iii) for each e, if R_e denotes the set of states of Q_e that are accessible from t_e^0 using exactly one transition of δ_e, then any state of R_e isn't accessible from any other state than t_e^0.

The assumption (iii) allows to characterize $\mathcal{T}$'s nodes that are associated by p with a selected node of $\mathscr{R}$ (i.e. with a node of I) as $\mathcal{T}$'s nodes that are associated, during a run of $\mathcal{A}_{\mathscr{R}}$, with a state of $\displaystyle\bigcup_{e=(i,j)\in M/j\in I} R_e$.

We denote by $Select(\mathcal{A}_{\mathscr{R}})$ this particular subset of states: $Select(\mathcal{A}_{\mathscr{R}}) = \displaystyle\bigcup_{e=(i,j)\in M/j\in I} R_e$

We give in the next subsection the formal construction of $\mathcal{A}_{\mathscr{R}}$.

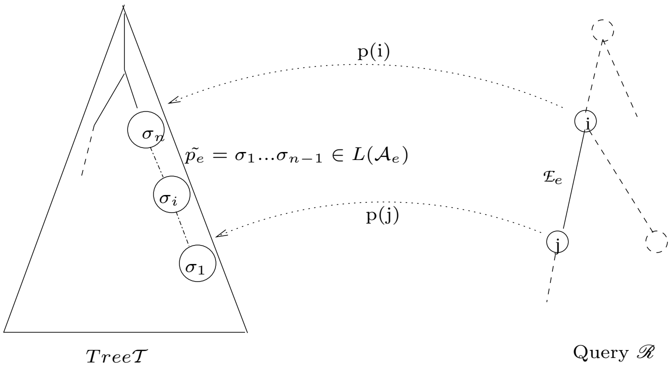


Figure 8: Underlying intuition of $\mathcal{A}_{\mathscr{R}}$'s construction

5.1.1 Construction of $\mathcal{A}_{\mathscr{R}} = (\Sigma, Q, \delta, F)$

Q (states) Let f and g be two states not occurring in $\cup_{e\in M} Q_e$, we define: $Q = \cup_{e\in M} Q_e \cup \{f, g\}$
F (Final states) $F = \{f\}$
δ (Transitions) Let x be a label in Σ and t a state in Q, we detail below the definition of $L(x, t) = \{w \in Q^*/(x, t, w) \in \delta\}$. We denote by τ the trace of $\mathscr{R}$ which is being recognized by some $\mathcal{A}_{\mathscr{R}}$'s run, and by p its associated $\mathscr{R}$'s mapping:

- If $t = g$, L(x,g)=g^*
  This transition set allows the automaton $\mathcal{A}_{\mathscr{R}}$ to associate a generic state g to a node n and to its descendants when n doesn't belong to the trace τ (transition 1 of Figure 9).

- If $t \in Q_e$ for some $e \in M$, L(x,t) is the union of three transition sets, $L(x, t) = L_1(x, t) \cup L_2(x, t) \cup L_3(x, t)$:

– The set $L_1(x,t)$ is non empty only when $e=(i,j)$ and j is a leaf node of $\mathscr{R}$. This transition set allows $\mathcal{A}_{\mathscr{R}}$ to start a $\mathcal{A}_e$'s run, from a leaf node of the trace τ, in order to recognize the word $\widetilde{p(e)}$.

$$L_1(x,t) = \begin{cases} g^* & \text{if j is a leaf node and (x, } t_e^0\text{, t)} \in \delta_e \\ \emptyset & \text{otherwise} \end{cases}$$

(transition 2 of Figure 9).

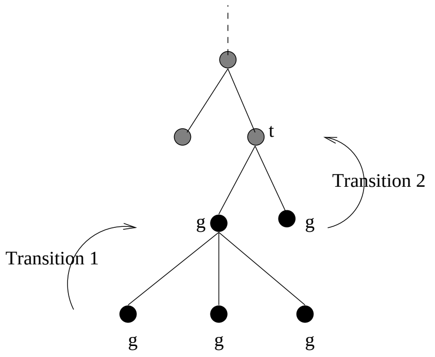


Figure 9: Transition1-2

– The set $L_2(x,t)$ allows $\mathcal{A}_{\mathscr{R}}$ to proceed further a $\mathcal{A}_e$'s run (transition 3 of Figure 10):

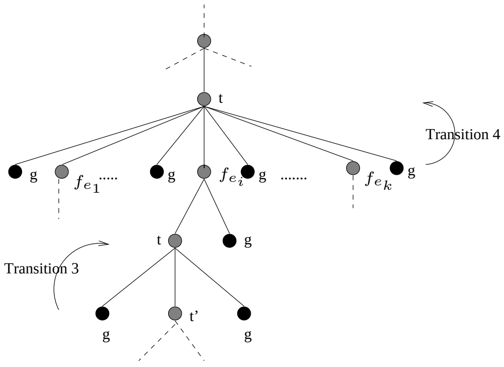


Figure 10: Transition 3-4

$$L_2(x,t) = \bigcup_{\{t'/(x,t',t)\in\delta_e\}} g^* t' g^*$$

– The set $L_3(x,t)$ allows $\mathcal{A}_{\mathscr{R}}$, when $e=(i,j)$ and $Out(j)=\{e_1,e_2,...,e_k\}$, to start a $\mathcal{A}_e$'s run at node j in order to recognize the word $\widetilde{p(e)}$ when the validations of the words $\widetilde{p(e_1)}...\widetilde{p(e_k)}$ have been successfully done:

$$L_3(x,t) = \begin{cases} g^* f_{e_1} g^* f_{e_2} g^* ... g^* f_{e_k} g^* & \text{if } j \text{ isn't a leaf in } \mathscr{R} \\ & \text{and } (x, t_e^0, t) \in \delta_e \\ \emptyset & \text{otherwise} \end{cases}$$

(transition 4 Figure 10)

- If $t=f$, $L(/,f) = g^* f_{e_1} g^*$ with $Out(1)=\{e_1\}$ in $\mathscr{R}$ and $L(x,f)=\emptyset$ if $x \neq /$
This transition set allows a run of $\mathcal{A}_{\mathscr{R}}$ to successfully stop at $\mathcal{T}$'s root.

5.1.2 Construction of $\bar{\mathcal{A}_{\mathscr{R}}}$=$(\Sigma, U, \eta, F)$

We now modify, for further needs, the automaton $\mathcal{A}_{\mathscr{R}}$ in order to obtain a new automaton $\bar{\mathcal{A}_{\mathscr{R}}}$ recognizing the same language as $\mathcal{A}_{\mathscr{R}}$ but providing an identification of the nodes of a $\mathscr{R}$'s trace that are descendant of a selected node. The idea is to slightly modify $\mathcal{A}_{\mathscr{R}}$ in order that successful runs associate overlined states to such nodes.
Formally, if $w=q_1...q_k$, let us denote by $\overline{w}$ the word $\overline{w} = \bar{q_1}...\bar{q_k}$ and $\overline{S} = \{\bar{w}/w \in S\}$ for any set of words S. We set $\bar{\mathcal{A}_{\mathscr{R}}}$=$(\Sigma, U, \eta, F)$ with $U = Q \cup \bar{Q}$ and we define the transition sets associated to η as follows:

- $L(x,\bar{g}) = \bar{g}^*$
- $L(x,\bar{t}) = \overline{L_1(x,t)} \cup \overline{L_2(x,t)} \cup \overline{L_3(x,t)}$
- $L(x,t) = \overline{L_1(x,t)} \cup \overline{L_3(x,t)}$ if $t \in R_e$, e=(i,j) and j ∈ I
- $L(x,t) = L_1(x,t) \cup L_2(x,t) \cup L_3(x,t)$ if $t \in Q_e$, e=(i,j) and j ∉ I

The two first transition sets associate overlined states to nodes that are descendant of selected nodes while the third one allows bottom-up $\bar{\mathcal{A}_{\mathscr{R}}}$'s runs to switch from overlined states to non overlined states when a selected node is reached.

Let us now notice that, for any mapping p of $\mathscr{R}$ on $\mathcal{T}$, nodes of $\mathcal{N}(trace_p(\mathcal{V},\mathcal{T})) \cup \mathcal{N}(\mathcal{V}_p(\mathcal{T}))$ are identified by the following property: they are associated, by a $\bar{\mathcal{A}_{\mathscr{R}}}$'s run on $\mathcal{T}$, to states of $U \setminus \{g\}$ i.e. to overlined states or to states of $\cup_{e\in M} Q_e \cup \{f\}$. This property will be used further.

5.2 Construction of $\mathcal{A}$

We now construct the automaton $\mathcal{A}$ from automata $\mathcal{A}_{\mathcal{C}}$ and $\bar{\mathcal{A}_{\mathcal{V}}}$, where $\mathcal{A}_{\mathcal{C}} = (\Sigma, Q_{\mathcal{C}}, \delta_{\mathcal{C}}, F_{\mathcal{C}})$ is the automaton built from the update query $\mathcal{C}$ using the construction of 5.1.1 and $\bar{\mathcal{A}_{\mathcal{V}}} = (\Sigma, U_{\mathcal{V}}, \eta_{\mathcal{V}}, F_{\mathcal{V}})$ is the automaton built from the view query $\mathcal{V}$ using the construction of 5.1.2. We use classical constructions for tree automata that we remember below.

Product automaton $\mathcal{A}_1 \times \mathcal{A}_2$
Let $\mathcal{A}_1 = (\Sigma, Q_1, \delta_1, F_1 \subseteq Q_1)$ and $\mathcal{A}_2 = (\Sigma, Q_2, \delta_2, F_2 \subseteq Q_2)$ be two tree automata. The language $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ is recognized by the product automaton $\mathcal{A}_1 \times \mathcal{A}_2$ defined by :
$\mathcal{A}_1 \times \mathcal{A}_2 = (\Sigma, Q_1 \times Q_2, \Delta, F_1 \times F_2)$
with $(x,(q,q'),(q_1,q_1')(q_2,q_2')\dots(q_n,q_n')) \in \Delta$
iff $(x,q,q_1q_2\dots q_n) \in \delta_1$ and $(x,q',q_1'q_2'\dots q_n') \in \delta_2$

Automaton with selective states $\sigma(\mathcal{B},S)$
Let us consider a tree automaton $\mathcal{B} = (\Sigma, U, \eta, F_U \subseteq U)$ and a subset of states $S \subseteq U$. The set of trees $\mathcal{T}$ on which there exists a run of $\mathcal{B}$ using at least one state of S, is a regular language. It is recognized by the automaton $\sigma(\mathcal{B},S)$ deduced from $\mathcal{B}$ as follows : roughly speaking $\sigma(\mathcal{B},S)$ works similarly to $\mathcal{B}$ except that it uses, besides U, a copy $\hat{U}$ of U. Once a node n is associated with a state of S, ascendant nodes of n are associated by a $\sigma(\mathcal{B},S)$'s run with states of $\hat{U}$ that are copies in $\hat{U}$ of the states they are associated with by a $\mathcal{B}$'s run. Formally:
we denote by π the mapping from $U \cup \hat{U}$ to U defined by

$\pi(\hat{u}) = \pi(u) = u$ for each $u \in U$ and we define
$\sigma(\mathcal{B}, S) = (\Sigma, U \cup \hat{U}, \eta \cup \gamma, \hat{F_U})$ with $\hat{F_U} = \{\hat{u}/u \in F_U\}$ and
$\gamma = \{(x, \hat{u}, w)/w \in (U \cup \hat{U})^* S (U \cup \hat{U})^*$ and $(x, u, \pi(w)) \in \eta\}$
$\cup \ \{(x, \hat{u}, w)/w \in (U \cup \hat{U})^* \hat{U} (U \cup \hat{U})^*$ and $(x, u, \pi(w)) \in \eta \ \}$.
The choice of $\hat{F_U}$ as final states ensures that each run of $\sigma(\mathcal{B}, S)$ uses at least one state of S.

From now on we suppose that a schema $\mathcal{S}c$ is available and specified by a finite automaton $\mathcal{A}_{\mathcal{S}c}$ i.e., $L(\mathcal{A}_{\mathcal{S}c}) = \mathbf{valid}(\mathcal{S}c)$

PROPOSITION 4. *$\mathcal{L} = L(\mathcal{A})$ where $\mathcal{A}$ is the automaton $\mathcal{A} = \mathcal{A}_{\mathcal{S}c} \times \sigma(\mathcal{A}_{\mathcal{C}} \times \bar{\mathcal{A}}_{\mathcal{V}}, S)$ with $S = Select(\mathcal{A}_{\mathcal{C}}) \times (U_{\mathcal{V}} \setminus \{g\})$.*

PROOF. Remember that the language $\mathcal{L}$ is the set of trees $\mathcal{T}$ satisfying the following conditions:
(i) $\mathcal{T} \in \texttt{valid}(\mathcal{S}c)$
(ii) There is a trace $trace_p(\mathcal{V}, \mathcal{T})$ of $\mathcal{V}$ in $\mathcal{T}$ with respect to some mapping p of $\mathcal{V}$ on $\mathcal{T}$,
(iii) There is a trace $trace_{p'}(\mathcal{C}, \mathcal{T})$ of $\mathcal{C}$ in $\mathcal{T}$ with respect to some mapping p' of $\mathcal{C}$ on $\mathcal{T}$,
(iv) $p'(I_c) \in \mathcal{N}(trace_p(\mathcal{V}, \mathcal{T})) \cup \mathcal{N}(\mathcal{V}_p(\mathcal{T}))$.
The set of trees satisfying conditions (ii) and (iii) is recognized by $\mathcal{A}_{\mathcal{C}} \times \bar{\mathcal{A}}_{\mathcal{V}}$. Moreover a run of $\mathcal{A}_{\mathcal{C}} \times \bar{\mathcal{A}}_{\mathcal{V}}$ on a tree $\mathcal{T}$ associates nodes of $p'(I_c) \cap \mathcal{N}(trace_p(\mathcal{V}, \mathcal{T})) \cup \mathcal{N}(\mathcal{V}_p(\mathcal{T}))$ with a state of $Select(\mathcal{A}_{\mathcal{C}}) \times (U_{\mathcal{V}} \setminus \{g\})$. Therefore the set of trees satisfying (ii), (iii) and (iv) is recognized by the automaton $\sigma(\mathcal{A}_{\mathcal{C}} \times \bar{\mathcal{A}}_{\mathcal{V}}, S)$ where $S = Select(\mathcal{A}_{\mathcal{C}}) \times (U_{\mathcal{V}} \setminus \{g\})$. Adding condition (i), we get $\mathcal{L} = L(\mathcal{A})$.

5.3 Complexity aspects

In this section we analyze the complexity of the construction of the automaton $\mathcal{A} = \mathcal{A}_{\mathcal{S}c} \times \sigma(\mathcal{A}_{\mathcal{C}} \times \bar{\mathcal{A}}_{\mathcal{V}}, S)$. We start with the complexity of the trace automaton construction.

LEMMA 1. *Let $\mathcal{A}_{\mathscr{R}} = (\Sigma, Q, \delta, F)$ be the automaton built in section 5.1.1 from the query $\mathscr{R} = (\Sigma, N, M, I, \mathscr{E})$ and let a_m be the maximal arity of N's nodes. The size $|\mathcal{A}_{\mathscr{R}}|$ of $\mathcal{A}_{\mathscr{R}}$ is in $O(|\Sigma| \times |\mathscr{R}| \times a_m)$.*

PROOF. Given $t \in Q$ and $x \in \Sigma$, let $\mathcal{A}_{L(x,t)}$ be a word automaton recognizing the language $L(x, t) = w \in Q^*/(x, t, w) \in \delta$. Following the construction of section 5.1.1, we have:

$$\begin{aligned}
|\mathcal{A}_{\mathscr{R}}| &= |Q| + \sum_{(x,t)\in\Sigma\times Q} |\mathcal{A}_{L(x,t)}| \\
&= |Q| + |\mathcal{A}_{L(/,f)}| + \sum_{x\in\Sigma} (|\mathcal{A}_{L(x,g)}|) \\
&\quad + \sum_{(x,t)\in\Sigma\times Q_M} (|\mathcal{A}_{L_1(x,t)}| + |\mathcal{A}_{L_2(x,t)}| + |\mathcal{A}_{L_3(x,t)}|)
\end{aligned}$$

where Q_M denotes $\cup_{e\in M} Q_e$.
Let us first recall that $|\mathscr{R}| = |N| + \sum_{e\in M} |\mathcal{A}_e|$ where $\mathcal{A}_e$ are automata associated to the regular expressions $\mathscr{E}_e$. Therefore $|Q| = |\bigcup_{e\in M} Q_e \cup \{f, g\}|$ is in $O(|\mathscr{R}|)$ and $\sum_{e\in M} |\delta_e| \leqslant |\mathscr{R}|$.
The result comes from the four following properties:

- $\forall e \in M$, $\forall (x, t) \in \Sigma \times Q_e, |\mathcal{A}_{L(x,g)}|$ and $|\mathcal{A}_{L_1(x,t)}|$ are in $O(1)$
  Hence $\sum_{x\in\Sigma} (|\mathcal{A}_{L(x,g)}|)$ is in $O(|\Sigma|)$ and $\sum_{(x,t)\in\Sigma\times Q_M} (|\mathcal{A}_{L_1(x,t)}|)$ is in $O(|\Sigma| \times |\mathscr{R}|)$
- There is a constant K such that: $\forall e \in M$, $\forall (x, t) \in \Sigma \times Q_e$, $|\mathcal{A}_{L_2(x,t)}| \leqslant K \times |\{t'/(x, t', t) \in \delta_e\}|$
  But $\sum_{(x,t)\in\Sigma\times Q_M} (|\mathcal{A}_{L_2(x,t)}|) = \sum_{e\in M} (\sum_{(x,t)\in\Sigma\times Q_e} (|\mathcal{A}_{L_2(x,t)}|))$.
  Hence $\sum_{(x,t)\in\Sigma\times Q_M} (|\mathcal{A}_{L_2(x,t)}|)$ is in $O(\sum_{e\in M} |\delta_e|)$
- $\forall e \in M$, $\forall (x, t) \in \Sigma \times Q_e, |\mathcal{A}_{L_3(x,t)}|$ is in $O(a_m)$ and there is at most $\sum_{e\in M} |\delta_e|$ pairs (x, t) such that $L_3(x, t)$ is not empty.
  Hence $\sum_{(x,t)\in\Sigma\times Q_M} (|\mathcal{A}_{L_3(x,t)}|)$ is in $O(a_m \times |\mathscr{R}|)$
- $|\mathcal{A}_{L(/,f)}|$ is in $O(1)$

LEMMA 2. *Let $\mathcal{B} = (\Sigma, U, \eta, F_U \subseteq U)$ be a tree automaton, a subset $S \subseteq U$ of states and $\Im = \sigma(\mathcal{B}, S)$ the automaton with selective states built in section 5.2. The size $|\Im|$ of $\Im$ is in $O(|\mathcal{B}|)$.*

PROOF. We have:
$|\Im| = 2|U| + \sum_{(x,\hat{u})\in\Sigma\times\hat{U}} |\mathcal{A}_{L_{\Im}(x,\hat{u})}| + \sum_{(x,u)\in\Sigma\times U} |\mathcal{A}_{L_{\Im}(x,u)}|$.
For each $(x, u) \in \Sigma \times U$, $L_{\Im}(x, u) = L_{\mathcal{B}}(x, u)$ and
$L_{\Im}(x, \hat{u}) = \pi^{-1}(L_{\mathcal{B}}(x, u)) \cap (U \cup \hat{U})^* (\hat{U} \cup S)(U \cup \hat{U})^*$.
So $|\mathcal{A}_{L_{\Im}(x,\hat{u})}|$ is in $O(|\mathcal{A}_{L_{\mathcal{B}}(x,u)}|)$.
Because $|\mathcal{B}| = |U| + \sum_{(x,u)\in\Sigma\times U} |\mathcal{A}_{L_{\mathcal{B}}(x,u)}|$, we deduce that $|\Im|$ is in $O(|\mathcal{B}|)$.

PROPOSITION 5. *The size $|\mathcal{A}|$ of the automaton $\mathcal{A} = \mathcal{A}_{\mathcal{S}c} \times \sigma(\mathcal{A}_{\mathcal{C}} \times \bar{\mathcal{A}}_{\mathcal{V}}, S)$ is in $O(a_{\mathcal{C}} a_{\mathcal{V}} \times |\Sigma|^2 \times |\mathcal{A}_{\mathcal{S}c}| \times |\mathcal{C}| \times |\mathcal{V}|)$, where $a_{\mathcal{C}}$ and $a_{\mathcal{V}}$ are the maximal arities of $\mathcal{C}$ and $\mathcal{V}$ respectively.*

PROOF. We have $|\mathcal{A}| \leqslant |\mathcal{A}_{\mathcal{S}c}| \times |\sigma(\mathcal{A}_{\mathcal{C}} \times \bar{\mathcal{A}}_{\mathcal{V}}, S)|$ and one easily deduces from the construction of section 5.1.2 that $|\bar{\mathcal{A}}_{\mathcal{V}}| \leq 2|\mathcal{A}_{\mathcal{V}}|$. Lemma 1 and 2 give then the result.

PROPOSITION 6. *The independence criterion $\mathcal{L} = \emptyset$ is polynomial and testable in $O(a_{\mathcal{C}}^2 \ a_{\mathcal{V}}^2 \ |\Sigma|^4 |\mathcal{A}_{\mathcal{S}c}|^2 \times |\mathcal{C}|^2 \times |\mathcal{V}|^2)$ time.*

PROOF. The standard algorithm for testing the emptiness of $\mathcal{A}$ that amounts to compute, by saturation until a fixpoint is reached, the subset $Acc \subseteq Q$ of accessible states, can be used. Its time complexity has been proved as quadratic in $|\mathcal{A}|$. The complexity of the independence criterion follows.

6. CONCLUSION

In this paper we have studied the problem of independence between views and updates. Our main contribution is a sufficient condition assuring the independence between a view query and a class of updates, testable in polynomial time. This condition is a necessary and sufficient condition in the case of linear view queries. We also showed that the problem of independence is in general PSPACE-hard.

For this study we chose a conceptual query language based on trees labeled by regular expressions. This language is quite general and includes some XPath fragments like tree patterns of $P^{*,//,[]}$ introduced in [15]. Our results can thus be applied to these fragments: an implementation of our independence test and an experimental study remain to be carried out, particularly in order to estimate how much time it saves to launch the independence test instead of evaluating the view query again.

Our analysis of independence between classes of updates and view queries brings down to an analysis of independence between two tree queries and could be used in other contexts: the problem of commutation between two update queries studied in [9] and in [5] is such an example. So we think that our approach is quite general and adaptable to application contexts requiring the analysis of relationships between several tree queries.

7. REFERENCES

[1] S. Abiteboul, J. McHugh, M. Rys, V. Vassalos, and J. L. Wiener. Incremental maintenance for materialized views over semistructured data. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 38–49, 1998.

[2] M. A. Ali, A. A. Fernandes, and N. W. Paton. Movie: An incremental maintenance system for materialized object views. *Data & Knowledge Engineering*, 47:131–166, 2003.

[3] A. Arion, V. Benzaken, I. Manolescu, and Y. Papakonstantinou. Structured materialized views for xml queries. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, 2007.

[4] A. Balmin, F. Özcan, K. S. Beyer, R. J. Cochrane, and H. Pirahesh. A framework for using materialized xpath views in xml query processing. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, 2004.

[5] M. Benedikt, A. Bonifati, S. Flesca, and A. Vyas. Verification of tree updates for optimization. In *CAV*, volume 3576, pages 379–393. Springer, 2005.

[6] J. A. Blakeley, P.-A. Larson, and F. W. Tompa. Efficiently updating materialized views. In *SIGMOD Conference*, pages 61–71, 1986.

[7] C. Byun, I. Yun, and S. Park. An efficient detection of conflicting updates in valid xml. In *CIT '07: Proceedings of the 7th IEEE International Conference on Computer and Information Technology*, Washington, DC, USA, 2007.

[8] K. Dimitrova, M. El-Sayed, and E. Rundensteiner. Order-sensitive view maintenance of materialized xquery views, 2003.

[9] G. Ghelli, K. H. Rose, and J. Siméon. Commutativity analysis in xml update languages. In *ICDT*, pages 374–388, 2007.

[10] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. pages 328–339, 1995.

[11] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applictions. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, 1995.

[12] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *ACM SIGMOD international conference on Management of data*, pages 157–166. ACM, 1993.

[13] L. V. S. Lakshmanan, H. Wang, and Z. Zhao. Answering tree pattern queries using views. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, 2006.

[14] H. Liefke and S. B. Davidson. View maintenance for hierarchical semistructured data. In *Data Warehousing and Knowledge Discovery*, pages 114–125, 2000.

[15] G. Miklau and D. Suciu. Containment and equivalence for a fragment of xpath. *ACM*, 51:2–45, 2004.

[16] M. Onizuka, F. Y. Chan, R. Michigami, and T. Honishi. Incremental maintenance for materialized xpath/xslt views.

[17] L. Quan, L. Chen, and E. Rundensteiner. Efficient refresh in an xql-based web caching system, 2000.

[18] M. Raghavachari and O. Shmueli. Conflicting xml updates. In *Advances in Database Technology - EDBT*, volume 3896, pages 552–569, 2006.

[19] A. Sawires, J. Tatemura, O. Po, D. Agrawal, and K. S. Candan. Incremental maintenance of path-expression views. In *ACM SIGMOD international conference on Management of data*, pages 443–454, 2005.

[20] M. H. Scholl, C. Laasch, and M. Tresch. Updatable views in object-oriented databases. In *Proc. 2nd Intl. Conf. on Deductive and Object-Oriented Databases (DOOD)*, number 566, 1991.

[21] D. Vista. *Optimizing Incremental View Maintenance Expressions In Relational Databases.* Thèse de doctorat, University of Toronto, 1996.

[22] W. C. XPath. Xml path language(xpath) version 1.0. Novembre 1999.

[23] Y. Zhuge and H. Garcia-Molina. Graph structured views and their incremental maintenance. In *Proc. 14th IEEE Conf. Data Engineering, ICDE*, pages 116–125. IEEE Computer Society, 1998.