Regular tree patterns: a uniform formalism for update queries and functional dependencies in XML *

Françoise Gire
CRI, Université Paris 1 Panthéon Sorbonne
90 rue de Tolbiac Paris, France
gire@univ-paris1.fr

Hicham Idabal
CRI, Université Paris 1 Panthéon Sorbonne
90 rue de Tolbiac Paris, France
hidabal@univ-paris1.fr

ABSTRACT

Given an XML functional dependency fd and a class of updates $\mathcal{U}$, we say that fd is independent with respect to $\mathcal{U}$ if and only if any XML document satisfies fd after any update q of $\mathcal{U}$, as soon as it did it before q. In this paper we study the following problem: is it possible to detect if an XML functional dependency fd is independent with respect to a class of updates $\mathcal{U}$? We address this problem when the functional dependency and the class of updates are specified with a same formalism: the *regular tree patterns*. We first show that the use of *regular tree patterns* federates most of the known approaches for expressing XML functional dependencies while allowing to capture some of constraints not so far expressible. Then we show that in general the addressed problem is PSPACE-hard, but we exhibit a sufficient condition testable in polynomial time ensuring the independence of a functional dependency with respect to a class of updates.

Keywords

XML, Functional dependency, Regular tree pattern, Update query

1. INTRODUCTION

Many topics have been intensively studied in the literature on XML: at the beginning schema definitions have been investigated to control the structural type of XML data, as well as query languages design to extract the desired information from these data. More recently, much effort has been done on (i) the definition of update languages to manage the inherent changing nature of XML data and on (ii) the enhancement of XML semantics expressiveness.

This last topic has becoming an important research topic and a lot of works can be found in the XML literature about integrity constraints ([7, 6]), general XML functional dependencies and more specifically XML keys ([4, 3, 5, 1, 16, 19, 17]). All these proposals differ the ones from the others depending on (a) they are or not independent from any schema such as DTDs or XSDs, (b) the ways they are accessing and comparing XML elements, and (c) their expressiveness and tractability. The challenge of most of these works has been to define specific classes of constraints (XML functional dependencies or XML key fragments) for which the implication problem is tractable, an axiomatization and normalization theory can be done, and efficient algorithms for constraint validation or reasoning about keys can be designed ([5, 13]). A clear survey about XML keys can be found in [12] as well as an interesting attempt in [8] to get an uniform formalism for expressing XML functional dependencies.

Because XML data can be frequently changed on the Web, another important issue regarding XML integrity constraints is the preservation of their validation on an XML document after one or more update operations. To our knowledge, very few works have addressed this issue in the literature: we can mention [5] where a method to maintain key indexes after an update is proposed and [14] where the impact of a set of updates on a set of XML functional dependencies has been investigated.

In this paper we study the preservation of the validation of a functional dependency on an XML document after a set of updates. We choose to express functional dependencies (FDs for short) as well as update queries through a same expressive formalism: the *regular tree patterns*. We show how this formalism federates most of previous approaches for XML FD designing and allows also to capture more kinds of constraints not so far expressible. In a previous work [9] we showed how the use of *regular tree patterns* for specifying view queries as well as update queries can be exploited to get an efficient algorithm detecting cases of independence of a view query with respect to a class of updates. We show in this paper that a similar algorithm can be designed to detect cases of independence of a set of FDs with respect to a class of updates, while this problem is in general PSPACE-hard. To our knowledge, only [14] has addressed the same problem. However in [14] the author supposes that the XML document is available, as well as extra stored information produced by previous FD verification passes. The uniforming notion of *regular tree pattern* allows us to get here a quite different approach dealing only with the FDs and update queries definitions and detecting non-violation

*Work supported by the ANR project ANR-08-DEFIS-04. Special thanks to the Orléans-Blois team of the project for its support to this work.

Updates in XML (EDBT Workshop Proceedings), March 22, 2010, Lausanne, Switzerland.

situations. More precisely, we exhibit a condition testable in polynomial time (in the sizes of the class of updates $\mathcal{U}$ and of the functional dependency fd) that ensures, for each XML document $\mathcal{D}$ satisfying fd, the non-violation of fd by $q(\mathcal{D})$ after any update q of $\mathcal{U}$.

The paper is organized as follows: Section 2 is devoted to preliminaries on XML and to the presentation of the *regular tree pattern* formalism. In Section 3 we show how *regular tree patterns* can be used to express functional dependencies on XML data. Section 4 is devoted to the use of *regular tree patterns* to define class of updates on XML documents. Finally in Section 5 we analyze the impact of a set of updates on some given FD when both are designed with *regular tree patterns*. We conclude in Section 6.

2. PRELIMINARIES

In this section we give our representation of XML documents and some preliminary definitions about the *regular tree pattern* formalism that we are using throughout the paper.

2.1 Representation of XML documents

XML documents are modelized by unranked ordered trees, labeled on some alphabet Σ (as shown on Figure 1) and we represent textual information such as attribute values or text contents by strings over another alphabet I. Then we consider that nodes of the document have one of the three types: *element node*(e), *attribute node*(a) or *text node*(t). Also we suppose that Σ is partitioned into three disjoint sets of labels, $\Sigma = El \cup A \cup \{S\}$, where labels from El label element nodes, labels from A label attribute nodes and S (indicating text) labels text nodes. Formally, a document $\mathcal{D}$ is a triplet $\mathcal{D}$=(D, λ, val) 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 $wj \in D, \forall j < i)$. Furthermore each internal node w of D has the type e and each leaf node of D has the type a or t. In the following, the domain D of $\mathcal{D}$=(D, λ, val) is also denoted by $\mathcal{N}(\mathcal{D})$.
- λ associates a label $l \in \Sigma$ with each node w of D according to its type as explained above. In particular $\lambda(w) \in A \cup \{S\}$ if and only if w is a leaf node of D.
- val is a valuation function from D to $D \cup I^*$ that is the identity on element nodes ($\forall w \in D$, $val(w) = w$ if and only if $\lambda(w) \in El$) and that associates a string value of I^* to each leaf node i.e. if and only if $\lambda(w) \in A \cup \{S\}$.

For technical reasons, we adopt the convention that the root ε is labeled with the symbol '/' of El in every document $\mathcal{D}$. Given two nodes w and w', a path p from w to w' is a sequence $p = (w_1, w_2, ..., w_n)$ of nodes such that: $w_1 = w$, $w_n = w'$ and $\forall i = 1, .., n-1$, $\exists j_i \in \mathbb{N}$ s.t. $w_{i+1} = w_i j_i$.

For each node w in $\mathcal{N}(\mathcal{D})$, we denote by $\mathcal{D}(w)$, the sub-tree rooted at w in $\mathcal{D}$ and defined by $\mathcal{D}(w) = (D_w, \lambda_w, val_w)$ with $D_w = \{wv / v \in \mathbb{N}^*\} \cap D$ and λ_w (respectively val_w) is the restriction of λ (respectively of val) to D_w. If $\overrightarrow{T}$ is a tuple $(\mathcal{D}(w_1), \mathcal{D}(w_2), ..., \mathcal{D}(w_n))$ of sub-trees of $\mathcal{D}$ then $\mathcal{N}(\overrightarrow{T})$ denotes the subset $D_{w_1} \cup D_{w_2} ... \cup D_{w_n}$ of $\mathcal{N}(\mathcal{D})$.

Figure 1 shows an XML document $\mathcal{D}$ storing data about an exam session. Each candidate is identified by its attribute @IDN: he has taken exams of different disciplines at different dates and has received for each of them a mark and a rank. Depending on the marks he received to the different exams, a global level (ranged from 'A' to 'E') is attributed to the candidate. Furthermore if the candidate didn't pass at least one exam, a child node *toBePassed* is added regrouping the remaining disciplines to be passed. On the contrary if all exams have been passed, the candidate is graduated and a child node *firstJob-Year* is added storing the year of his first job.

2.2 Regular tree patterns

We present here preliminary notions about *regular tree patterns* that we need later. Intuitively, an n-ary regular tree pattern is a formalism to select in an XML document, tuples of nodes satisfying a particular condition. Roughly speaking this condition requires the existence, in the document, of a tree-shaped sub-structure (a so-called *trace* of the regular tree pattern) conforming to a particular template and carrying the tuple of nodes to be selected.

If Σ is a finite alphabet, $REG(\Sigma)$ denotes the set of regular expressions over Σ. A regular expression is said proper if and only if its associated regular language of words does not contain the empty word ϵ.

DEFINITION 1 (N-ARY REGULAR TREE PATTERN). *Let Σ be a finite alphabet of labels. An n-ary regular tree pattern over Σ is defined by $\mathcal{R} = (\mathcal{T}, \overrightarrow{s})$ where*

- *$\mathcal{T} = (\Sigma, N, E, \mathcal{E})$ is the regular tree template of $\mathcal{R}$ composed of:*
  - a tree (N, E) where N is a tree domain and $E \subseteq N \times N$ is the set of edges given by the natural child relation on words of $\mathbb{N}^*$.
  - an application $\mathcal{E} : E \longrightarrow REG(\Sigma)$ which associates each edge (w, w') of E with a proper regular expression of $REG(\Sigma)$, denoted by $\mathcal{E}_{(w,w')}$
- *$\overrightarrow{s} = (w_1, ..., w_n)$ is the selected tuple of $\mathcal{R}$ composed of nodes of N. When $n = 1$, $\mathcal{R}$ is called a* monadic regular tree pattern.

The size of $\mathcal{R}$ denoted by $|\mathcal{R}|$ is defined by: $|\mathcal{R}| = |N| + \Sigma_{e \in E} |\mathcal{A}_e|$ where $\mathcal{A}_e$ is a word automaton associated to the regular expression $\mathcal{E}_e$ and $|\mathcal{A}_e|$ denotes the size of $\mathcal{A}_e$.

The evaluation of an n-ary regular tree pattern on an XML document $\mathcal{D}$ consists in (a) identifying in $\mathcal{D}$ all tuples of nodes that satisfy the conditions expressed by the tree pattern and (b) returning the tuples of subtrees in $\mathcal{D}$ rooted at these nodes. Formally the identifying phase (a) of the

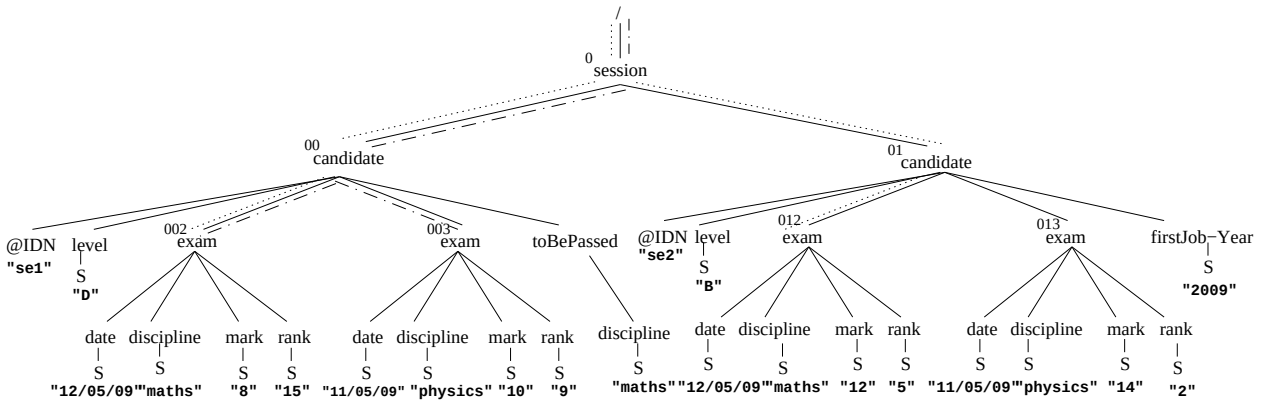


Figure 1: An XML document

evaluation uses the concept of mapping. In the next definition "<" denotes the natural document order (equivalently descendant or following order) between two nodes of a document $\mathcal{D}$.

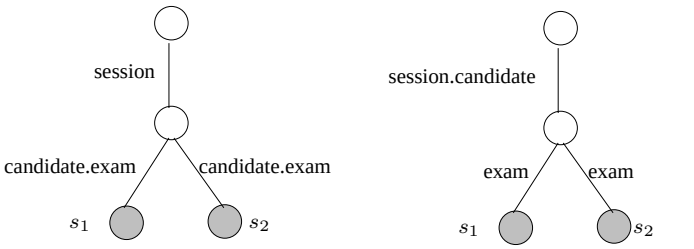
DEFINITION 2 (MAPPING OF A REGULAR TREE PATTERN). *A mapping on an XML document $\mathcal{D}$ of an n-ary regular tree pattern $\mathcal{R} = (\mathcal{T}, \vec{s})$ where $\mathcal{T} = (\Sigma, N, E, \mathcal{E})$, is an application π from N to $\mathcal{N}(\mathcal{D})$ such that:*

- *If r is the root node of $\mathcal{T}$ then $\pi(r)$ is the root node of $\mathcal{D}$ labeled with '/'*
- *$\forall w, w' \in N$, if $w < w'$ then $\pi(w) < \pi(w')$*
- *$\forall e = (w, w') \in E$, there exists in $\mathcal{D}$ a path π_e from $\pi(w)$ to $\pi(w')$, such that:*
  (a) the sequence, denoted by $\lambda(\pi_e)$, of the concatenated labels occurring on π_e ($\lambda(\pi(w))$ excluded and $\lambda(\pi(w'))$ included) is a word of the language associated to $\mathcal{E}_e$ and,
  (b) if $e_1 = (w, w_1)$ and $e_2 = (w, w_2)$ are two distinct outgoing edges from a node w of N then paths π_{e_1} and π_{e_2} must not have common prefixes in $\mathcal{D}$.

Trace of a regular tree pattern with respect to a mapping The trace of the regular tree pattern $\mathcal{R}$ in $\mathcal{D}$ with respect to the mapping π is the smallest sub-tree of $\mathcal{D}$ containing the image $\pi(N)$. We denote it by $\texttt{trace}_\pi(\mathcal{R}, \mathcal{D})$.

Evaluation of a regular tree pattern on a document Let $\mathcal{P}$ be the set of all mappings of the regular tree pattern $\mathcal{R}$ on the document $\mathcal{D}$.

- The result, denoted by $\overrightarrow{\mathcal{R}_\pi(\mathcal{D})}$, of the evaluation of $\mathcal{R}$ on $\mathcal{D}$ with respect to the mapping π of $\mathcal{P}$, is the tuple of sub-trees $\overrightarrow{\mathcal{R}_\pi(\mathcal{D})} = (\mathcal{D}(\pi(w_1)), \ldots, \mathcal{D}(\pi(w_n)))$ where $\vec{s} = (w_1, ..., w_n)$ is the selected tuple of $\mathcal{R}$.
- The result of the evaluation of $\mathcal{R}$ on $\mathcal{D}$ is denoted $\mathcal{R}(\mathcal{D})$ and defined by: $\mathcal{R}(\mathcal{D}) = \bigcup_{\pi \in \mathcal{P}} \overrightarrow{\mathcal{R}_\pi(\mathcal{D})}$

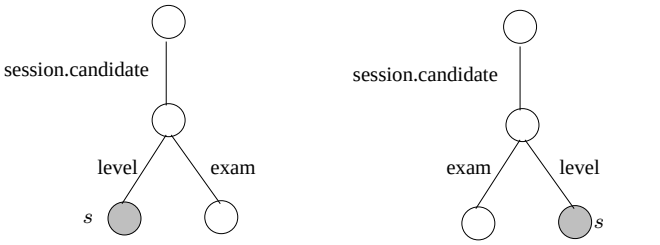


Regular tree pattern $\mathcal{R}_1$ Regular tree pattern $\mathcal{R}_2$

Figure 2: Regular Tree Patterns

Let us consider the regular tree patterns $\mathcal{R}_1 = (\mathcal{T}_1, (s_1, s_2))$ and $\mathcal{R}_2 = (\mathcal{T}_2, (s_1, s_2))$ of Figure 2 where the selected nodes s_1 and s_2 are grayed. Their evaluations on document $\mathcal{D}$ of Figure 1 are quite different. On the one hand, because of condition (b) of Definition 2, the meaning of $\mathcal{R}_1$ is "*Select pairs of sub-trees rooted at exam nodes taken by two different candidates*". Therefore there are four mappings of $\mathcal{R}_1$ on $\mathcal{D}$ and four pairs selected by $\mathcal{R}_1$ on $\mathcal{D}$: $\mathcal{R}_1(\mathcal{D}) = \{(\mathcal{D}(002), \mathcal{D}(012)), (\mathcal{D}(002), \mathcal{D}(013)), (\mathcal{D}(003), \mathcal{D}(012)), (\mathcal{D}(003), \mathcal{D}(013))\}$. On the other hand, the meaning of $\mathcal{R}_2$ is "*Select pairs of sub-trees rooted at exam nodes taken by a same candidate*". So there are only two mappings of $\mathcal{R}_2$ on $\mathcal{D}$ and two pairs selected by $\mathcal{R}_2$ on $\mathcal{D}$: $\mathcal{R}_2(\mathcal{D}) = \{(\mathcal{D}(002), \mathcal{D}(003)), (\mathcal{D}(012), \mathcal{D}(013))\}$.
A trace of $\mathcal{R}_1$ (respectively $\mathcal{R}_2$) can be seen on Figure 1 drawn with a dotted (respectively dashed) line.

Let us also notice that a mapping of a regular tree pattern $\mathcal{R}$ must respect the order of the nodes in $\mathcal{R}$. As illustration, the evaluation of the regular tree pattern $\mathcal{R}_3 = (\mathcal{T}_3, (s))$ of Figure 3 on document $\mathcal{D}$ of Figure 1 returns "*Sub-trees rooted at level nodes for the candidates having passed at least one exam*" while the evaluation of the regular tree pattern $\mathcal{R}_4 = (\mathcal{T}_4, (s))$ of Figure 3 on the same document $\mathcal{D}$ is empty.



Regular tree pattern $\mathcal{R}_3$ Regular tree pattern $\mathcal{R}_4$

Figure 3: Regular Tree Patterns

3. FUNCTIONAL DEPENDENCIES AS REGULAR TREE PATTERNS

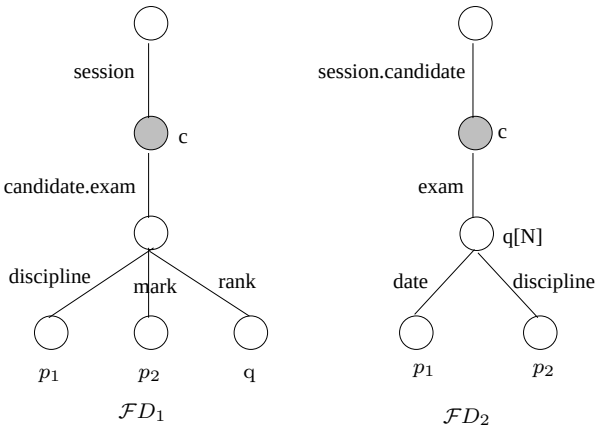
3.1 Example 1

Let us consider the XML document of Figure 1 and the functional dependency fd_1: *In a session, two exams concerning the same discipline and evaluated with the same mark, share the same rank value.*

We express the functional dependency fd_1 with the pair $(\mathcal{F}D_1, c)$ where $\mathcal{F}D_1 = (\mathcal{T}_1, \overrightarrow{s_1} = (p_1, p_2, q))$ is the regular tree pattern of Figure 4, and the nodes c, p_1, p_2 and q of $\mathcal{N}(\mathcal{T}_1)$ have the following meaning:
- c (grayed on Figure 4) is the context node under which fd_1 must be verified
- nodes p_1 and p_2 of $\overrightarrow{s_1}$ are the condition nodes
- node q of $\overrightarrow{s_1}$ is the target node

Semantically a document $\mathcal{D}$ satisfies fd_1 if and only if, for two different traces of $\mathcal{F}D_1$ in $\mathcal{D}$, $\tau_1 = \texttt{trace}_{\pi_1}(\mathcal{F}D_1, \mathcal{D})$ and $\tau_2 = \texttt{trace}_{\pi_2}(\mathcal{F}D_1, \mathcal{D})$, that are in the same session ($\pi_1(c)$ and $\pi_2(c)$ are the same node) and whose exams concern the same discipline and are evaluated with the same mark ($\pi_1(p_1)$ and $\pi_2(p_1)$, as well as $\pi_1(p_2)$ and $\pi_2(p_2)$, share the same value), the attributed ranks coincide ($\pi_1(q)$ and $\pi_2(q)$ share the same value too).

This example shows how regular tree patterns are well adapted to express functional dependencies. We give next a formal definition of XML functional dependencies using regular tree patterns.



$\mathcal{F}D_1$ $\mathcal{F}D_2$

Figure 4: XML functional dependencies

3.2 Functional dependencies on XML documents

In the previous example we used two kinds of equality for nodes: a *node equality* (identification) for the context node and a *value equality* to compare the condition and target nodes. We precise below these notions.

DEFINITION 3 (VALUE EQUALITY). *Two nodes, w_1 and w_2, of a document $\mathcal{D}$=(D, λ, val) are* value-equal *if and only if:*
- $\lambda(w_1) = \lambda(w_2)$ (they have the same label)
- w_1 and w_2 have the same type τ
- if τ is a *(attribute type) or* t *(text type), then $val(w_1) = val(w_2)$*
- if τ is e *(element type), then $|\{w_1 i / w_1 i \in D \wedge i \in \mathbb{N}\}| = |\{w_2 i / w_2 i \in D \wedge i \in N\}|$ and for each $w_1 i$ in D, $w_1 i$ and $w_2 i$ are value-equal*

We use the following notations:
- $w_1 =_V w_2$ if w_1 and w_2 are *value-equal*
- $w_1 =_N w_2$ if w_1 and w_2 are the same node

Many proposals for defining XML functional dependencies can be found in the literature ([1, 16, 19, 17]). However we think that the use of regular tree patterns (a) makes uniform most of the main proposals in the sense that it allows to concisely and precisely define the relationship between the specifications of the condition and target nodes and (b) enhances their expressive power as we will see later through some examples.

DEFINITION 4. *(XML functional dependency) An XML functional dependency is an expression $fd = (\mathcal{F}D, c)$ where:*

- *$\mathcal{F}D = (\mathcal{T}, \overrightarrow{s} = \{p_1[E_1], p_2[E_2], ..., p_n[E_n], q[E_{n+1}]\})$ is a regular tree pattern whose each selected node $p_1, ..., p_n$ and q is associated with an equality type $E_i \in \{V, N\}$ (i=1,..., n+1)*
- *c is an ancestor node of each node $p_1, p_2, ..., p_n$ and q*

c is the context node, the p_i's are the condition nodes and q is the target node

For sake of simplicity, when omitted, the equality types E_is are set by default to V. So we write p_i instead of $p_i[V]$.

DEFINITION 5. *(Satisfaction of a functional dependency) A document $\mathcal{D}$ satisfies the functional dependency $(\mathcal{F}D, c)$ if and only if for two traces, $\tau_1 = \texttt{trace}_{\pi_1}(\mathcal{F}D, \mathcal{D})$ and $\tau_2 = \texttt{trace}_{\pi_2}(\mathcal{F}D, \mathcal{D})$, of $\mathcal{F}D$ in $\mathcal{D}$ satisfying*
(a) $\pi_1(c) =_N \pi_2(c)$ (the context node images are the same) and (b) $\forall i = 1, ..., n$, $\pi_1(p_i) =_{E_i} \pi_2(p_i)$,
the equality $\pi_1(q) =_{E_{n+1}} \pi_2(q)$ holds as well.

Example 2: Let us consider the functional dependency fd_2 for the XML document of Figure 1: *A candidate can not take at the same date two different exams concerning the same discipline*
In our formalism, $fd_2 = (\mathcal{F}D_2, c)$ where $\mathcal{F}D_2$ is the regular tree pattern of Figure 4. The context node c is grayed. When equal to N, the equality type is written next to the associated condition or target node.

Also in [8], the authors propose a formalism to make uniform most of the previous proposals for expressing XML functional dependencies. So to compare our approach with the previous proposals, we choose to compare it with those of [8]. Following the formalism of [8] a functional dependency fd is an expression with the form
$(C, (\{P_1[E_1], P_2[E_2], ..., P_n[E_n]\} \rightarrow Q[E_{n+1}]))$: C is an absolute simple linear path specifying the context node,
$P_1[E_1], ..., P_n[E_n]$ and $Q[E_{n+1}]$ are simple linear paths relative to the context node and specify the condition and target nodes of fd with their equality types. As examples, the functional dependencies fd_1 and fd_2 are expressed in [8] with the following expressions $expr_1$ and $expr_2$:
$expr_1$: (/session, ({candidate/exam/discipline, candidate/exam/mark } → candidate/exam/rank))
$expr_2$: (/session/candidate, ({exam/date, exam/discipline } → exam[N]))
The semantics associated in [8] to $(C, (\{P_1[E_1], P_2[E_2], ..., P_n[E_n]\} \rightarrow Q[E_{n+1}]))$ is mainly the same as those we associate to $(\mathcal{F}D, c)$ where $\mathcal{F}D = (\mathcal{T}, (p_1, p_2, ..., p_n, q))$ is the regular tree pattern built from the paths $C, P_1, ..., P_n, Q$ as follows: roughly speaking, the construction of $\mathcal{F}D$ translates the paths C, P_1, ...,P_n, Q into words w_c, w_{p_1},...,w_{p_n},w_q of labels of Σ, and uses these words to label edges in the template $\mathcal{T}$; w_c labels an edge from the root node to the context node c, and the words w_{p_1},...,w_{p_n} and w_q label paths from the context node c to the condition and target nodes $p_1, p_2, ..., p_n$ and q. Furthermore a particular policy is followed that introduces additional nodes between the context node c and the p_is or q nodes, to factorize when it exists, the longest common prefix between two any words of $\{w_{p_1}, w_{p_2}, ..., w_{p_n}, w_q\}$. For example, the regular tree patterns built from $expr_1$ and $expr_2$ give back the regular tree patterns $\mathcal{F}D_1$ and $\mathcal{F}D_2$ of Figure 4.
The only difference between the semantics associated in [8] to $(C, (\{P_1[E_1], P_2[E_2], ..., P_n[E_n]\} \rightarrow Q[E_{n+1}]))$ and the semantics we associate to $(\mathcal{F}D, c)$ is the fact that no order is required in [8] between the paths $P_1, ..., P_n, Q$ while our semantics requires each mapping of $\mathcal{F}D$ to respect the order between sibling nodes of $\mathcal{T}$.
Thus the regular tree pattern formalism allows to express the functional dependencies expressed with the formalism of [8], adding however node ordering requirements. It also allows us to express more kinds of functional dependencies as shown by the next example.

Example 3: Here we suppose that, in the documents we are working on, the 'exam' nodes of each candidate are sorted by discipline and each candidate has at least one exam in a given discipline. Given such documents, we want to express the following functional dependencies:
(fd_3) *Two candidates with the same mark in at least two disciplines receive the same level*
(fd_4) *Two candidates with the same mark in at least two disciplines and also having some remaining exams to pass, receive the same level*

In our formalism, $fd_3 = (\mathcal{F}D_3, c)$ and $fd_4 = (\mathcal{F}D_4, c)$ where $\mathcal{F}D_3$ and $\mathcal{F}D_4$ are the regular tree patterns of Figure 5. However neither fd_3 nor fd_4 can be expressed by the formalism of [8]. Indeed, the regular tree pattern $\mathcal{F}D$ built from $(C, (\{P_1[E_1], P_2[E_2], ..., P_n[E_n]\} \rightarrow Q[E_{n+1}]))$ with the above detailed construction, satisfies particular properties that keep it from expressing such dependencies:

- Labels of two edges outgoing from a same node can not have a common prefix. Therefore fd_3 can not be expressed by such a pattern. On the contrary, fd_3 can be expressed with the regular tree pattern $\mathcal{F}D_3$ thanks to its sibling edges labeled by *exam* and thanks to condition (b) of Definition 2: indeed condition (b) ensures that any mapping captures marks from two different exams taken by a same candidate.
- Leaves of $\mathcal{F}D$ are only condition or target nodes. Therefore fd_4 that requires the existence of another leaf node (namely a node labeled by *toBePassed*) can not be expressed by such a pattern. On the contrary, fd_4 is clearly expressed by the regular tree pattern $\mathcal{F}D_4$ of Figure 5.

Finally regular tree patterns are a very simple and concise formalism to express XML functional dependencies supporting most of the previous proposals. Furthermore they enhance their expressive power allowing to precise ordering requirements and to express more complex dependencies thanks to the use of (a) regular expressions as labels and (b) general tree templates.

4. XML UPDATE CLASSES AS REGULAR TREE PATTERNS

As already mentioned, the formalism of *n-ary regular tree pattern* allows to select from an XML document, tuples of nodes satisfying particular conditions. Therefore this formalism is also well-adapted to query XML documents. In a previous work [10] we showed that, as an XML query language, the *regular tree pattern* formalism is incomparable (regarding the expressive power) with Full XPATH as defined in [18] . However it can express all queries of the positive fragment of CoreXPATH ([11]).
Regular tree patterns are also well adapted to express updates on XML documents. Indeed an update on an XML document $\mathcal{D}$ consists in (1) selecting a set of nodes in $\mathcal{D}$ to be updated and (2) replacing the sub-tree $\mathcal{D}(w)$ rooted at each selected node w by a new sub-tree. This modelling 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 an update q of an XML document is defined in this paper as the composition $q = u \circ \mathcal{U}$ of two applications u and $\mathcal{U}$: application $\mathcal{U}$ selects the set of nodes w to be updated and application u performs the updates by replacement of the sub-trees $\mathcal{D}(w)$ rooted at these

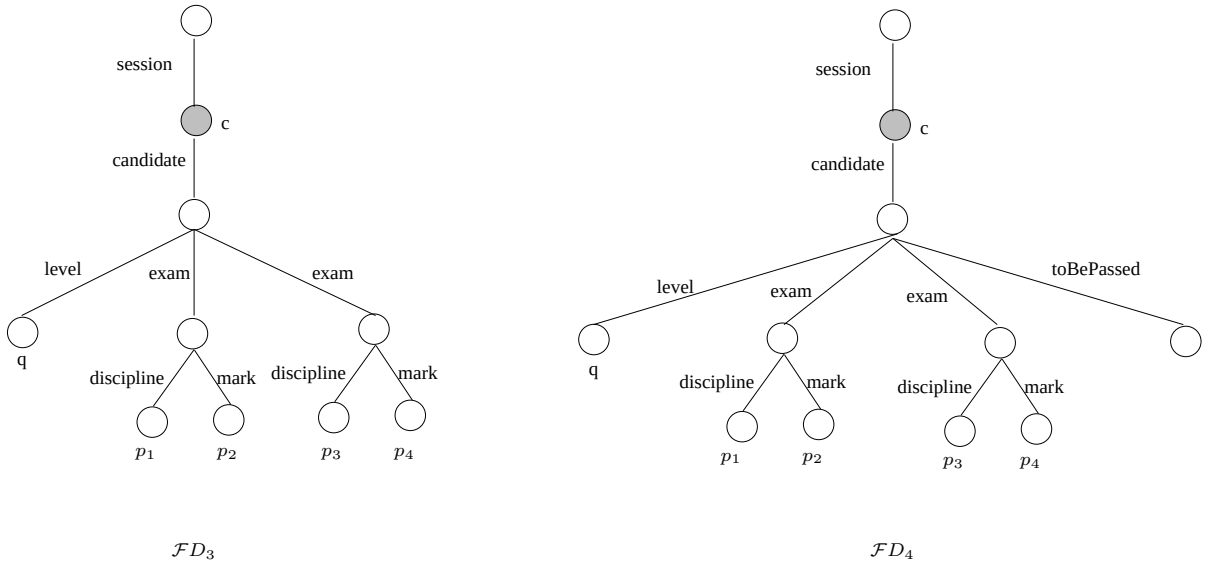


Figure 5: More XML functional dependencies

nodes w. Application $\mathcal{U}$ represents in fact a class of updates: two updates belong to the same class $\mathcal{U}$ if and only if they are defined with the same node selecting application $\mathcal{U}$. Therefore a class of updates $\mathcal{U}$ can be expressed through a regular tree pattern that only defines the set of nodes to be updated without performing the precise updates at these nodes.

Example 4 The regular tree pattern $\mathcal{U} = (\mathcal{T}_\mathcal{U}, s_\mathcal{U})$ of Figure 6 defines a class of updates that update levels of candidates still having to pass some remaining exams. Here only one node ($s_\mathcal{U}$) is selected to be updated. The update queries q_1: "*For each candidate still having to pass some remaining exams, decrease his level to the level just below*" and q_2: "*For each candidate still having to pass some remaining exams, add a child node 'comment' to the 'level' node*" are two updates belonging to the class $\mathcal{U}$. When evaluated on the document $\mathcal{D}$ of Figure 1, the class $\mathcal{U}$ returns only the node '001' to be updated because there is only one mapping of $\mathcal{U}$ on $\mathcal{D}$.

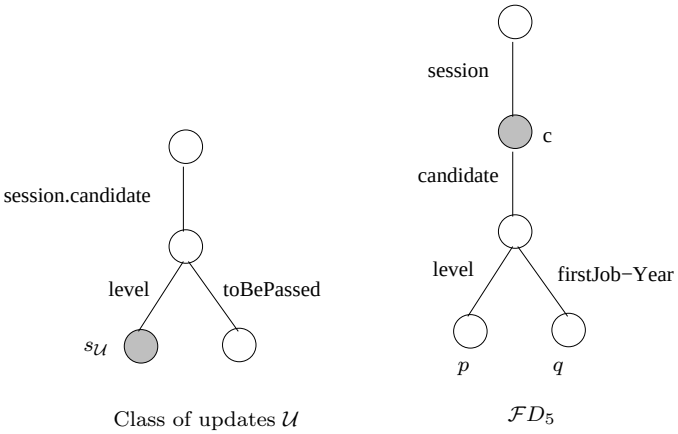


Figure 6: Updates and FDs

5. INDEPENDENCE BETWEEN UPDATES AND FUNCTIONAL DEPENDENCIES

Impact We say that an update q has an impact on an XML functional dependency fd if and only if there is a document $\mathcal{D}$ such that $\mathcal{D}$ satisfies fd and $q(\mathcal{D})$ doesn't satisfy fd.

Example 5: The update q_1 of Example 4 has an impact on the functional dependency fd_3. Indeed a document $\mathcal{D}$ satisfying fd_3 can contain two candidates γ_1 and γ_2 with the same marks in at least two disciplines and the same level but γ_1 has a child *toBePassed* while γ_2 not: so q_1 only updates γ_1's level and fd_3 is becoming violated in $q_1(\mathcal{D})$.

Update-FD Independence Our goal is to analyze both the functional dependency fd and the update query q in order to detect an eventual impact of q on fd. It is important to notice here that our objective is to detect this impact independently of any source document that is not supposed available during the analysis. In addition, in order to simplify the problem, we suppose that the detailed specification of the function u, performing the update, is unknown and therefore that u can be of any type. Hence we choose to focus on the detection of impacts of a whole class $\mathcal{U}$ of updates (rather than of a particular update q) on a functional dependency. The *Update-FD independence problem* is thus formally stated as follows: *the functional dependency fd is independent with respect to a class of updates $\mathcal{U}$ if and only if for every update q in $\mathcal{U}$, q doesn't impact fd.*

Update-FD Independence in the context of a schema In many cases, the availability of a schema constraining the source documents can improve the independence analysis. Let us denote by $\texttt{valid}(\mathcal{S}c)$ the set of valid documents with respect to a schema $\mathcal{S}c$. In this context the independence definition is modified as follows: *the functional dependency*

fd is independent with respect to a class of updates $\mathcal{U}$ in the context of the schema $\mathcal{S}c$ if and only if: $\forall\ \mathcal{D} \in \texttt{valid}(\mathcal{S}c)$, $\forall\ q \in \mathcal{U}$ with $q(\mathcal{D}) \in \texttt{valid}(\mathcal{S}c)$, if $\mathcal{D}$ satisfies fd then $q(\mathcal{D})$ satisfies fd as well.

Example 6: Let us consider Figure 6 and suppose we are in the context of a schema $\mathcal{S}c$ that requires each candidate to have a child *toBePassed* or a child *firstJob-Year* but not both. Then $fd_5 = (\mathcal{F}D_5, c)$ is independent with respect to the class of updates $\mathcal{U}$ (where $\mathcal{U}$ and $\mathcal{F}D_5$ are shown on Figure 6): indeed any update of $\mathcal{U}$ modifies levels only of candidates having a child *toBePassed* and therefore, that are not concerned by fd_5.

We show now how the choice of a uniform formalism to express functional dependencies and classes of updates can be useful in the analysis of the independence problem of a functional dependency with respect to a class of updates.

Let $fd = (\mathcal{F}D, c)$ be an XML functional dependency with context node c and tree pattern
$\mathcal{F}D = (\mathcal{T}, \overrightarrow{s} = (p_1[E_1], p_2[E_2], ..., p_n[E_n], q[E_{n+1}]))$.
Let $\mathcal{U} = (\mathcal{T}_\mathcal{U}, \overrightarrow{s}_\mathcal{U})$ be an update class. We restrict our approach to update classes expressed by regular tree patterns whose updated nodes (nodes of $\mathcal{N}(\overrightarrow{s}_\mathcal{U})$) are leaves of $\mathcal{T}_\mathcal{U}$. As we will see later, this condition allows us to get a polynomial sufficient criterion ensuring the independence of fd with respect to $\mathcal{U}$.
Finally let $\mathcal{S}c$ be a schema requiring some structure constraints on XML documents. In this work $\mathcal{S}c$ is supposed to be given by some regular Bottom-Up tree automaton $\mathcal{A}_{\mathcal{S}c}$.
We start by giving in the next section a general complexity result about the *Update-FD independence* problem.

5.1 PSPACE-hardness

PROPOSITION 1. *Deciding whether a functional dependency fd is independent with respect to a class of updates $\mathcal{U}$ 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 the label alphabet $\Sigma = \{A, B, C, D, F, G, \#\}$ and two expressions η and η' of $REG(\Sigma)$ where label '#' does not occur. We define the two regular tree patterns $\mathcal{F}D$ and $\mathcal{U} = (\mathcal{T}_\mathcal{U}, s_\mathcal{U})$ as in Figure 7 and prove that $fd = (\mathcal{F}D, c)$ is impacted by $\mathcal{U}$ iff $\eta \not\subseteq \eta'$.
Suppose that fd is impacted by $\mathcal{U}$. There exists a document $\mathcal{D}$ satisfying fd and an update $q \in \mathcal{U}$ such that $q(\mathcal{D})$ does not satisfy fd . Therefore there are:
- two mappings π_1 and π_2 of $\mathcal{F}D$ on $q(\mathcal{D})$ whose traces $\tau^1_{\mathcal{F}D}$ and $\tau^2_{\mathcal{F}D}$ are witnesses of the violation of fd in $q(\mathcal{D})$,
- a mapping π of $\mathcal{U}$ on $\mathcal{D}$, whose trace τ selects the updated node: notice that τ remains on $q(\mathcal{D})$ because $s_\mathcal{U}$ is a leaf of $\mathcal{T}_\mathcal{U}$.
Now if $\pi(\mu)$ is distinct from both $\pi_1(\nu)$ and $\pi_2(\nu)$ in $q(\mathcal{D})$ (where μ and ν are the nodes shown on Figure 7) then $\tau^1_{\mathcal{F}D}$ and $\tau^2_{\mathcal{F}D}$ would already be witnesses of the violation of fd in $\mathcal{D}$. So let us suppose $\pi(\mu) = \pi_1(\nu)$: then q does not modify any node of $\tau^2_{\mathcal{F}D}$ nor ascendants or descendants nodes of $\pi_1(p)$ and $\pi_1(q)$. Therefore, if $\eta \subseteq \eta'$, a witness of the violation of fd in $\mathcal{D}$ can be obtained from τ and $\tau^2_{\mathcal{F}D}$.

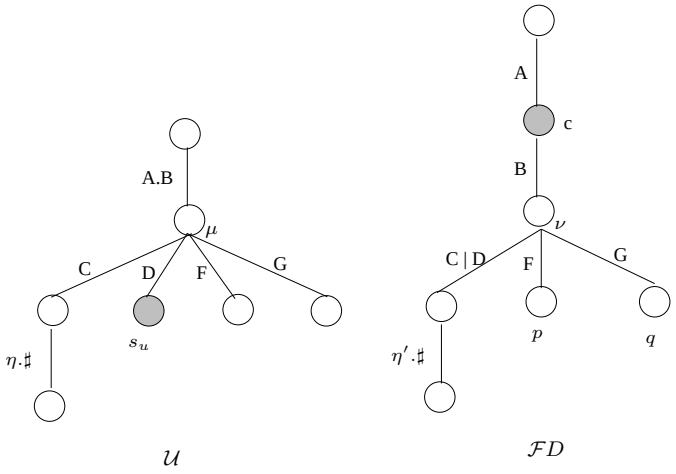


Figure 7: Reduction schema

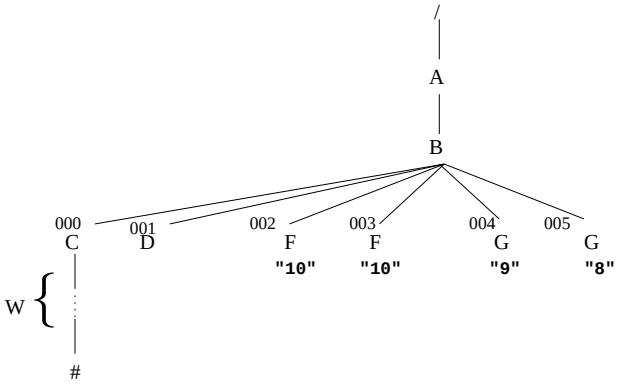


Figure 8: Witness of dependency

Conversely if $\eta \not\subseteq \eta'$, we consider the document $\mathcal{D}$ shown on Figure 8 where the sequence of labels encountered on the path from the 'C' node to the '$\#$' node is a word w of $L(\eta)$ that does not belong to $L(\eta')$. Because w does not belong to $L(\eta')$, $\mathcal{D}$ satisfies fd. Now because w belongs to $L(\eta)$, node 001 is updated by any update of $\mathcal{U}$. Let us consider an update q of $\mathcal{U}$ that adds a descendant path $w'\#$ to the node 001, where w' is a word of $L(\eta')$. Now clearly $q(\mathcal{D})$ does not satisfy fd because the two nodes labeled by G have different values while the ones labeled by F are value-equal.

□

5.2 An independence criterion

We define a language $\mathcal{L}$ of XML documents satisfying particular conditions, and we establish a relationship between the vacuity of $\mathcal{L}$ and the independence of fd with respect to $\mathcal{U}$ in the context of $\mathcal{S}c$.

DEFINITION 6. *Let $\mathcal{L}$ be the language of XML documents $\mathcal{D}$ satisfying:*
(i) $\mathcal{D} \in \texttt{valid}(\mathcal{S}c)$
(ii) There is in $\mathcal{D}$ a trace of $\mathcal{F}D$, $\tau_{\mathcal{F}D} = trace_\pi(\mathcal{F}D, \mathcal{D})$,

with respect to a mapping π of $\mathcal{FD}$ on $\mathcal{D}$,
and there is in $\mathcal{D}$ a trace of $\mathcal{U}$, $\tau_{\mathcal{U}} = trace_{\pi'}(\mathcal{U}, \mathcal{D})$, with respect to a mapping π' of $\mathcal{U}$ on $\mathcal{D}$, such that:
$\mathcal{N}(\pi'(\overrightarrow{s}_{\mathcal{U}})) \bigcap (\mathcal{N}(trace_{\pi}(\mathcal{FD}, \mathcal{D})) \cup \mathcal{N}(\mathcal{FD}_{\pi}(\mathcal{D}))) \neq \emptyset$

Roughly speaking XML documents of $\mathcal{L}$ are $\mathcal{S}c$-valid documents that contain a trace $\tau_{\mathcal{FD}}$ of $\mathcal{FD}$ and a trace $\tau_{\mathcal{U}}$ of $\mathcal{U}$ such that there is at least a selected node of $\tau_{\mathcal{U}}$ (a node of $\mathcal{N}(\pi'(\overrightarrow{s}_{\mathcal{U}}))$) that is a node of $\tau_{\mathcal{FD}}$ (i.e. of $\mathcal{N}(trace_{\pi}(\mathcal{FD}, \mathcal{D}))$ or a node of the sub-trees rooted at condition or target nodes of fd (i.e. nodes of $\mathcal{N}(\mathcal{FD}_{\pi}(\mathcal{D}))$ selected by $\mathcal{FD}$).

PROPOSITION 2 (INDEPENDENCE CRITERION *IC*). *If $\mathcal{L}$ is empty then fd is independent with respect to $\mathcal{U}$ in the context of $\mathcal{S}c$.*

PROOF. Suppose that fd is not independent with respect to $\mathcal{U}$ in the context of $\mathcal{S}c$, then there is an update q of $\mathcal{U}$, a document $\mathcal{D}$ of $\texttt{valid}(\mathcal{S}c)$ such that $q(\mathcal{D})$ is also in $\texttt{valid}(\mathcal{S}c)$ and, $\mathcal{D}$ satisfies fd while $q(\mathcal{D})$ does not satisfy fd. Therefore there is at least a node n updated by q whose update generates a witness of the violation of fd in $q(\mathcal{D})$. Two cases can occur:

- (a) The violation of fd in $q(\mathcal{D})$ involves a trace $\tau_{\mathcal{FD}}$ of $\mathcal{FD}$ in $q(\mathcal{D})$, that already was in $\mathcal{D}$ but has not been touched by q ($\tau_{\mathcal{FD}} = trace_{\pi}(\mathcal{FD}, \mathcal{D})$). Thus necessarily n is a node of one of the sub-trees rooted at condition or target nodes and its update modifies the values of this sub-tree generating the violation of fd in $q(\mathcal{D})$ (case 1 Figure 9). So n is a node of $\mathcal{N}(\pi'(\overrightarrow{s}_{\mathcal{U}})) \cap \mathcal{N}(\mathcal{FD}_{\pi}(\mathcal{D}))$ and $\mathcal{D}$ is a document of $\mathcal{L}$.
- (b) The update of n creates a new trace $\tau_{\mathcal{FD}}$ of $\mathcal{FD}$ in $q(\mathcal{D})$ (with respect to some mapping π), that is a witness of the violation of fd in $q(\mathcal{D})$: therefore n is, in $q(\mathcal{D})$, a node of $\tau_{\mathcal{FD}}$ (case 2 Figure 9). Now because any node of $\mathcal{N}(\overrightarrow{s}_{\mathcal{U}})$ is a leaf of $\mathcal{T}_{\mathcal{U}}$, the trace of $\mathcal{U}$ in $\mathcal{D}$ carrying n, remains in $q(\mathcal{D})$: so n is also a node of a trace $\tau_{\mathcal{U}}$ of $\mathcal{U}$ in $q(\mathcal{D})$ (with respect to some mapping π'). Finally n belongs to $\mathcal{N}(\pi'(\overrightarrow{s}_{\mathcal{U}})) \cap \mathcal{N}(trace_{\pi}(\mathcal{FD}, q(\mathcal{D})))$ and $q(\mathcal{D})$ is a document of $\mathcal{L}$.

□

We now give a method to check the independence criterion *IC*.

PROPOSITION 3 (CHECKING CRITERION *IC*). *Given an XML functional dependency $fd = (\mathcal{FD}, c)$, a class of updates $\mathcal{U} = (\mathcal{T}_{\mathcal{U}}, \overrightarrow{s}_{\mathcal{U}})$ and a regular Bottom-Up automaton $\mathcal{A}_{\mathcal{S}c}$ specifying a schema $\mathcal{S}c$,*

- *A regular Bottom-Up automaton $\mathcal{A}$ recognizing $\mathcal{L}$ can be built from the automaton $\mathcal{A}_{\mathcal{S}c}$ and the regular tree patterns $\mathcal{FD}$ and $\mathcal{U}$*

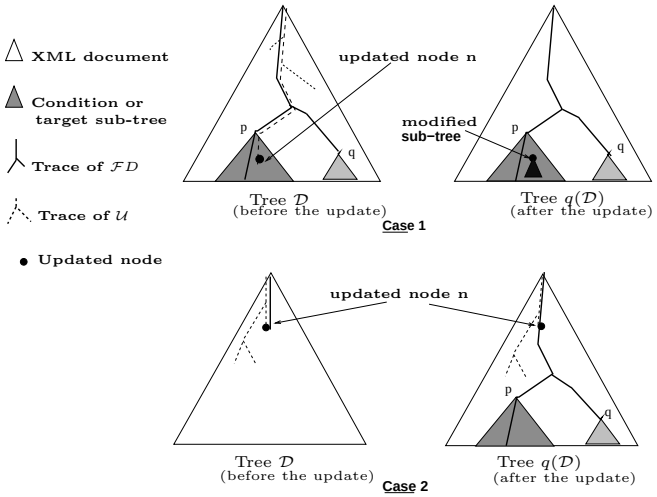


Figure 9: Impact analysis

- *The size $|\mathcal{A}|$ of the automaton $\mathcal{A}$ is in $O(a_{\mathcal{U}} a_{\mathcal{FD}} \times |\Sigma|^2 \times |\mathcal{A}_{\mathcal{S}c}| \times |\mathcal{U}| \times |\mathcal{FD}|)$, where $a_{\mathcal{U}}$ and $a_{\mathcal{FD}}$ are the maximal arities of $\mathcal{U}$ and $\mathcal{FD}$ respectively*
- *The independence criterion IC is polynomial: more precisely, the emptiness of the language $\mathcal{L}$ is testable in $O(a_{\mathcal{U}}^2\ a_{\mathcal{FD}}^2 \times |\Sigma|^4 \times |\mathcal{A}_{\mathcal{S}c}|^2 \times |\mathcal{U}|^2 \times |\mathcal{FD}|^2)$ time.*

Sketch of the proof: For sake of space, we only give here an idea of the construction of the automaton $\mathcal{A}$. Firstly, given a regular tree pattern $\mathcal{R}$ we define an automaton $\mathcal{A}_{\mathcal{R}}$ that recognizes the language of documents containing a trace of $\mathcal{R}$. Secondly, given the functional dependency $fd = (\mathcal{FD}, c)$ and the class of updates $\mathcal{U} = (\mathcal{T}_{\mathcal{U}}, \overrightarrow{s}_{\mathcal{U}})$, we combine the automata $\mathcal{A}_{\mathcal{FD}}$ and $\mathcal{A}_{\mathcal{U}}$ built from the regular patterns $\mathcal{FD}$ and $\mathcal{U}$ respectively, in order to get another automaton $\mathcal{B}$ recognizing the language of documents satisfying conditions (ii) of Definition 6. Finally, the automaton $\mathcal{A}$ is built as a product automaton between the automata $\mathcal{A}_{\mathcal{S}c}$ and $\mathcal{B}$.

Related work

The problem of independence between XML functional dependencies and classes of updates is very close to the one of independence between XML views and updates, that has already been studied in different previous works [15, 2, 9]. In [9] we also used regular tree patterns to define views and updates on XML documents and adopted the same approach as here to analyze their independence. So our contribution here is to show how the regular tree pattern formalism, that is independent from any standard, is well adapted to express XML functional dependencies and is also useful for discovering cases of independence of XML FDs with respect to classes of updates as soon as both are expressed with regular tree patterns.

To our knowledge only the work of [14] has already addressed the problem of XML functional dependencies verification after a set of updates. The approach followed in [14] is however quite different from ours: (a) firstly it uses the source document as well as additional information stored from previous

verification passes on the document, (b) secondly the set of updates is given through a set of positions in the source document and finally, (c) the updates to apply are completely detailed for each selected node.
Therefore when the independence criterion *IC* is satisfied our solution is more efficient because it only uses the definitions of FDs, of the class of updates and possibly of a schema: it avoids the whole parsing of the source document performed in [14] that can be expensive in case of huge documents.
However when the independence criterion *IC* is not satisfied, we can not conclude about an eventual impact of the class of updates on the functional dependency, while the algorithm of [14] is more precise and concludes in any case using the source document, the additional stored information and the details of the performed updates.
Finally another difference with the work of [14] is our ability to treat more complex FDs on XML documents thanks to the use of regular tree patterns.

6. CONCLUSION

In this paper we proposed to express functional dependencies on XML documents with the regular tree pattern formalism. We showed how this formalism can federate most of the previous approaches for XML functional dependency designing while capturing other kinds of constraints not expressible so far.

Also we showed that, as soon as classes of updates are specified with regular tree patterns too, it is possible to define a sufficient criterion testable in polynomial time that ensures the independence of a functional dependency fd with respect to a class $\mathcal{U}$ of updates (possibly in presence of a schema). However, the problem of independence is in general PSPACE-hard. In [10] we showed that the *regular tree pattern* formalism can express all queries of the positive fragment of CoreXPath. Our results can thus be applied when the classes of updates are specified with positive queries of CoreXPath.

Obviously we are aware that the choice of *regular tree patterns* to express functional dependencies brings to a class of XML constraints whose associated axiomatisation and verification problems remain to be studied but are probably untractable in general. However we showed that, for some problems like the one addressed in this paper, efficient algorithms can be designed to detect particular situations where answers can be found. The PSPACE-hardness of the update-FD independence does not give an upper bound for its complexity. We conjecture that the problem is in PSPACE but a precise proof remains to be given.

Finally an implementation of our independence criterion and an experimental study are of course still missing and remain to be carried out, particularly in order to estimate how much time it saves to launch the independence criterion instead of verifying the functional dependency again.

7. REFERENCES

[1] M. Arenas and L. Libkin. A normal form for XML documents. *ACM Trans. Database Syst.*, 29(1):195–232, 2004.

[2] M. Benedikt and J. Cheney. Schema-based independence analysis for XML updates. In *VLDB '09: Proceedings of the 35th International Conference on Very Large Data Bases*, 2009.

[3] P. Buneman, S. Davidson, W. Fan, C. Hara, and W.-C. Tan. Reasoning about keys for XML. *Inf. Syst.*, 28(8):1037–1063, 2003.

[4] P. Buneman, S. B. Davidson, W. Fan, C. S. Hara, and W. C. Tan. Reasoning about keys for XML. In *DBPL '01: Revised Papers from the 8th International Workshop on Database Programming Languages*, pages 133–148, London, UK, 2002. Springer-Verlag.

[5] Y. Chen, S. B. Davidson, and Y. Zheng. Xkvalidator: a constraint validator for XML. In *CIKM '02*, pages 446–452, New York, NY, USA, 2002. ACM.

[6] W. Fan and L. Libkin. On XML integrity constraints in the presence of dtds. *J. ACM*, 49(3):368–406, 2002.

[7] W. Fan and J. Siméon. Integrity constraints for XML. In *PODS '00*, pages 23–34, New York, NY, USA, 2000. ACM.

[8] M. Ferrari, M. Lima, and B. Bouchou. Contraintes d'intégrité pour XML. visite guidée par une syntaxe homogène. In *Revue des sciences et technologies de l'information*, pages 331–364. Hermes science publications, 2007.

[9] F. Gire and H. Idabal. Updates and views dependencies in semi-structured databases. In *IDEAS '08: Proceedings of the 2008 international symposium on Database engineering & applications*, pages 159–168, New York, NY, USA, 2008. ACM.

[10] F. Gire and H. Idabal. Requêtes arbres régulières pour l'analyse de dépendances entre vues et mises à jour de documents XML. *To appear in RTSI-ISI:Revue des Sciences et Technologies de l'Information - Serie Ingénierie des Systèmes d'Information*, 2010.

[11] P. H. Hartel. A trace semantics for positive core xpath. *Temporal Representation and Reasoning, International Syposium on*, 0:103–112, 2005.

[12] S. Hartmann, H. Köhler, S. Link, T. Trinh, and J. Wang. On the notion of an XML key. In *SDKB 2008*, volume 4925, pages 114–123. LNCS - Springer, 2008.

[13] S. Hartmann and S. Link. Unlocking keys for XML trees. In *ICDT 2007*, volume 4353/2006, pages 104–118. LNCS - Springer, 2007.

[14] M. A. Lima. *Maintenance incrémentale des contraintes d'intégrité en XML*. Phd, Université François Rabelais de Tours, France, 2007.

[15] M. Raghavachari and O. Shmueli. Conflicting XML updates. In *EDBT*, volume 3896, pages 552–569, 2006.

[16] M. W. Vincent, J. Liu, and C. Liu. Strong functional dependencies and their application to normal forms in XML. *ACM Trans. Database Syst.*, 29(3):445–462, 2004.

[17] J. Wang. A comparative study of functional dependencies for XML. In *APWeb*, pages 308–319, 2005.

[18] W. C. XPath. XML path language(XPath) version 1.0. Novembre 1999.

[19] C. Yu and H. V. Jagadish. Efficient discovery of XML data redundancies. In *VLDB '06*, pages 103–114. VLDB Endowment, 2006.