# Maximal Set of XML Functional Dependencies for the Integration of Multiple Systems

Joshua Amavi(✉) and Mirian Halfeld Ferrari

INSA Centre Val de Loire, Univ. Orléans, LIFO EA 4022 FR-45067, Orléans, France
{joshua.amavi,mirian}@univ-orleans.fr

**Abstract.** A web application expected to deal with XML documents conceived on the basis of divers sets of (local) constraints would be expected to test documents with respect to all non contradictory constraints imposed by these original (local) sources. The goal of this paper is to introduce an optimized algorithm for computing the maximal set of XML functional dependencies (XFD) over multiple systems. The basis of our method is a sound and complete axiom system which is provided for relative XFD allowing two kinds of equality: value or node equality.

**Keywords:** XML · Functional dependencies · XFD · Interoperability

## 1 Introduction

This paper deals with the problem of exchanging XML (eXtensible Markup Language) data in a multi-system environment where a global central system should receive and process data coming from different local sources. Our global system is a conservative evolution of local ones. It conserves the possibility of accepting XML documents coming from any local (original) source. It extends local systems since it has its own schema and integrity constraints (generated from a merge of local ones) and may accept and deal with non-local XML documents (possibly non locally valid ones).

Our work aims at enriching schema evolution proposals by taking into account integrity constraints. Schema merging proposals are usually based on simple data models. Schemas can be more expressive than DTD and XSD, associated to integrity constraints (as in [6]) or expressed by a semantically richer data model (as in [23]).

A conservative schema evolution algorithm that extends minimally regular tree grammar is proposed in [9]. That approach for schema extension is inherently syntactic: only structural aspects of XML documents are considered and new grammars are built by syntactic manipulation of the original production rules. This paper aims at enriching that model by offering the possibility of computing from given local sets of XFD (XML Functional Dependencies), a cover of the biggest set of XFD that does not violate any local document. This is a first

Partially supported by CAPES-Stic/AmSud Project 052/2014 (SWANS).

step towards an extension of a schema evolution proposal which will take into account integrity constraints. This extension intends to enrich schema evolution but is conceived as an independent procedure. In this way it may be applied or adapted to other schema evolution approaches.

Some applications of our work are: in the field of Digital Libraries, due to their need of evolution when new sources of data become available or when merging two libraries may be interesting [11]; in the construction of innovative services with data coming from diverse organizations that manipulate similar (though not identical) information, allowing us to envisage possible adaptations to big data applications [7]. In these cases, it is important to have a non contradictory set of integrity constraints (one that could be built from the original local constraints).

We suppose that $S_1, \dots, S_n$ are local (original) systems which deal with sets of XML documents $X_1, \dots, X_n$, respectively, and that inter-operate with a global, integrated system $S$. System $S$ integrates local systems and is seen as an evolution of all of them. It can continue to receive information from any local (original) system, but it can also deal with information coming from other non local sources. Each set $X_i$ conforms to schema constraints $\mathcal{D}_i$ and to integrity constraints $\mathcal{F}_i$ and follows an ontology $O_i$. Our goal is to associate system $S$ to type and integrity constraints which represent a conservative evolution of local constraints. More precisely, given different triples $(\mathcal{D}_1, \mathcal{F}_1, O_1), \dots, (\mathcal{D}_n, \mathcal{F}_n, O_n)$, we are interested in generating $(\mathcal{D}, cover\mathcal{F}, \mathcal{A})$, where:
$(i)$ $\mathcal{D}$ is an extended type that accepts any local document;
$(ii)$ $cover\mathcal{F}$ is a set of XFD equivalent to $\mathcal{F}$ the biggest set of functional dependencies (XFD), built from $\mathcal{F}_1, \dots, \mathcal{F}_n$, that can be satisfied by all documents in $X_1, \dots, X_n$ and
$(iii)$ $\mathcal{A}$ is an ontology alignment that guides the construction of $\mathcal{D}$ and $\mathcal{F}$ in terms of semantics mapping. Notice that ontology issues are out of the scope of this paper, but we suppose the existence of $\mathcal{A}$ which is the basis of a pre-processing step where correspondence among tree paths (built on the different $D_i$) is established. The construction of this pre-processor is out of purpose in this paper; we just consider that the output of such pre-processing is an input of our algorithms.

This paper focus only on the generation of $cover\mathcal{F}$ which contains the XFD for which no violation is possible when considering document sets $X_1, \dots, X_n$. It is important to notice that our algorithm is based on an axiom system and, thus, obtains $cover\mathcal{F}$ from $\mathcal{F}_1, \dots, \mathcal{F}_n$, disregarding data.

The contribution of this paper is twofold. On one hand we introduce an axiom system together with the proofs of its soundness and completeness. On the other hand, we present an efficient way for computing, on the basis of our axiom system, the set $cover\mathcal{F}$. We prove that the obtained set $cover\mathcal{F}$ has good properties and some experiments show the efficiency of our approach.

The rest of this paper is organized as follows. Section 2 comments on some related work. Section 3 illustrates our goal with an example. Section 4 presents some background while Section 5 introduces our XFD. Section 6 focuses on our axiom system. Section 7 introduces our method for computing *cover*$\mathcal{F}$ while in Section 8 we discuss on some experiments. Finally, Section 9 concludes the paper. We refer to [2] for the omitted proofs.

## 2 Related Work

Our motivation is to offer to a *global* system the capability of preserving the biggest set of local non contradictory constraints. Since the objective is to work only on constraint specification without any data involvement, we use an axiom system. To the best of our knowledge no other work considers this scenario.

We refer to [3,13,16,19–21,24] as other proposals for defining XFD and to [13,22] for a comparison among some of them. Different XFD proposals entail different axiomatisation system, such as those in [13,15,21]. We adopt XFD presented in [4] for which we possess a validation tool (general algorithm in [6]). The approach in [12] defines XFD as tree queries, which implies a complex implementation, and proposes static XFD validation *w.r.t.* updates.

To achieve our goal out first task is to propose an axiom system for the adopted XFD, together with an efficient algorithm for computing the closure of a set of paths. Our work on this axiom system is comparable to the one proposed in [21]. The main differences are: (i) we propose a more powerful path language allowing the use of a wild-card; (ii) our XFD are verified *w.r.t.* a context and not only *w.r.t.* the root, *i.e.*, XFD can be relative; (iii) our XFD can be defined by taking into account two types of equality: value and node equality and (iv) we use simpler concepts (such as branching paths, projection) which, we believe, allow us to prove that our axiom system is sound and complete in a clearer way.

We use our axiom system in the development of a practical tool: to filter local XFD in order to obtain a set containing only XFD that cannot be violated by any local XML document. Our global system aims to deal with data coming from any local source, but not to perform data fusion. Thus, our work presents an original point of view, since we are not interested in putting together all the local information, but just in manipulating them. Usually, schema integration proposal comes together with the idea of data fusion. XML data fusion is considered in papers such as [8,18]. Data exchange is considered in [10] that aims to construct an instance over a target schema, based on the source and a given mapping, and to answer queries against the target data, consistently with the source data.

Proposals concerning XML type evolution usually do not take into account the evolution of associated integrity constraints which are extremely important in the maintenance of consistent information. In [14] authors offer as a perspective to apply to XML their proposal of adapting functional dependencies according to schema changes. This is done in [19] where authors consider the problem of constraint evolution in conformance with type evolution. The type evolution in [9] is well adapted to our purposes; it seems possible to combine it with our XFD filter in order to generate a set of constraints allowing interoperability.

## 3 Motivating Example

We suppose universities or educational institutions, from the same region in France, which want to implement a central data system to obtain and process information concerning their courses and students, independently of local systems already in use. Their goal is to obtain a central system that ensures a maximum number of the local non-contradictory integrity constraints.

Each educational institution has established, locally, its own constraints. For instance, let the XML trees in Figure 1 be documents from two different universities. Each document is valid *w.r.t.* the functional dependencies presented in Table 1, *i.e.*, documents in $X_1$ are valid *w.r.t.* $\mathcal{F}_1$, those in $X_2$ are valid *w.r.t.* $\mathcal{F}_2$. We recall that local schemas and concepts may be different.

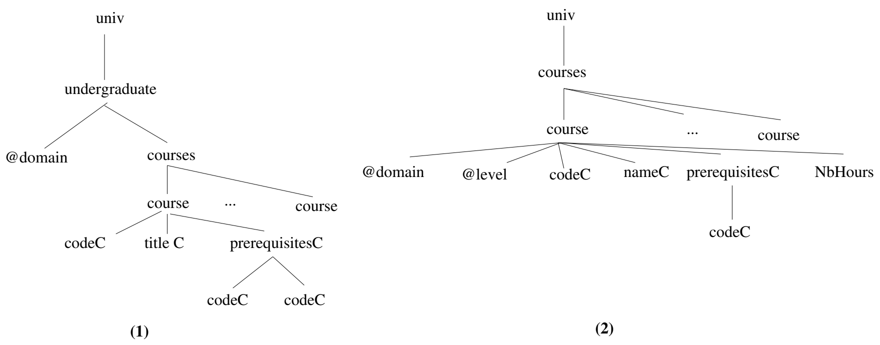

**Fig. 1.** Two XML documents from different local sources

In the XML domain, a functional dependency (XFD) is defined by paths over a tree. Each path selects a node on a tree. Values or positions of the selected nodes are gathered to build tuples that will be used to verify whether a given XML document satisfies an XFD. For example, consider the XFD *f: (univ, (undergraduate/courses/course/codeC* $\rightarrow$ *undergraduate/courses/course/titleC))* on the first document of Figure 1(1). It specifies that the context is *univ*, *i.e.*, that the constraint should be verified on data below a node labelled *univ*. In this context, *f* entails the construction of tuples composed by values obtained by following the paths: *univ/undergraduate/courses/course/codeC, univ/undergraduate/-courses/course/titleC.* As in the relational model, a document is valid *w.r.t. f* if any two tuples agreeing on values obtained from *univ/undergraduate/courses/course/codeC* also agree on values obtained from *univ/undergraduate/courses/course/titleC.* Thus, in a university the code of a course determines its name.

Similarly, the XFD *f1: (univ, (undergraduate/courses/course/codeC* $\rightarrow$ *undergraduate/courses/course/prerequisitesC))* entails tuples where the path *univ/undergraduate/courses/course/prerequisitesC* leads us to obtain sub-trees having roots labelled *prerequisitesC* (*i.e.*, sub-trees containing information about prerequisites). This constraint indicates that courses having the same code should have the same prerequisites. A document is valid *w.r.t.* $f1$ if any two tuples agreeing on values obtained from *univ/undergraduate/courses/course/codeC* also agree on values obtained from *univ/undergraduate/courses/course/prerequisitesC*, *i.e.*, obtained sub-trees are isomorphic.

**Table 1.** XFD in $\mathcal{F}_1$ and $\mathcal{F}_2$

| $\mathcal{F}$ | XFD |
|---|---|
| 1 | $(univ, (undergraduate/courses/course/codeC \rightarrow undergraduate/courses/course/titleC))$ |
| 1 | $(univ, (undergraduate/courses/course/codeC \rightarrow undergraduate/courses/course/prerequisitesC))$ |
| 1 | $(univ, (undergraduate/courses/course/codeC \rightarrow undergraduate/@domain))$ |
| 2 | $(univ, (courses/course/codeC \rightarrow courses/course/nameC))$ |
| 2 | $(univ, (courses/course/codeC \rightarrow courses/course/@domain))$ |
| 2 | $(univ, (courses/course/codeC \rightarrow courses/course/@level))$ |
| 2 | $(univ, (\{courses/course/nameC, courses/course/@level\} \rightarrow courses/course/NbHours))$ |

Now consider the first three XFD in Table 1, concerning source 1. They indicate that in a university, the code of a course determines its name, its domain and its prerequisites. In other words, a course is identified by its code.

From the alignment of local ontologies we assume that Table 2 is available, making the correspondence among paths on the different local sets of documents. Thus, it is possible to conclude that, for instance, XFD *f: (univ, (undergraduate/courses/course/codeC → undergraduate/courses/course/titleC))* and *(univ, (courses/course/codeC → courses/course/nameC))* are equivalent, *i.e.*, they represent the same constraint since they involve the same concepts: in a university, the code of a course determines its name.

**Table 2.** Extract of the translation table

| Paths from $\mathcal{D}_1$ | Paths from $\mathcal{D}_2$ |
|---|---|
| *univ/undergraduate/courses/course/codeC* | *univ/courses/course/codeC* |
| *univ/undergraduate/courses/course/titleC* | *univ/courses/course/nameC* |
| *univ/undergraduate/courses/course/prerequisitesC* | *univ/courses/course/prerequisitesC* |
| *univ/undergraduate/courses/course/prerequisitesC/codeC* | *univ/courses/course/prerequisitesC/codeC* |
| *univ/undergraduate/@domain* | *univ/courses/course/@domain* |

Assuming that we have only these two local sources, we want to obtain, from $\mathcal{F}_1$ and $\mathcal{F}_2$, the biggest set of XFD $\mathcal{F}$ that does not contradict any document in $X_1$ and $X_2$. To reach this goal, we should consider *all* XFD derivable from $\mathcal{F}_1$ and $\mathcal{F}_2$, which may result in very big sets of XFD. Indeed, the set $\mathcal{F}$ is, usually, a very big one - too big to work with. A better solution consists of computing *cover*$\mathcal{F}$, *a cover of* $\mathcal{F}$ (*i.e.*, a (usually) smaller set of XFD that is equivalent to $\mathcal{F}$), without computing *all* XFD derivable from $\mathcal{F}_1$ and $\mathcal{F}_2$. In this paper, we propose an algorithm that generates this set of XFD.

In our example, the resulting *cover*$\mathcal{F}$ would contain XFD of Table 3. Let us analyse this solution. In Table 3, the first and the fourth XFD are equivalent. They are kept in *cover*$\mathcal{F}$ since all documents in $X_1$ and $X_2$ are valid *w.r.t.* it. The same reasoning is applied for the second and third XFD in Table 3. The two last XFD involve concepts that occur only in $X_2$ and, thus, cannot be violated by documents

**Table 3.** XFD in the resulting $\mathcal{F}$

| | |
|---|---|
| 1 | $(univ, (undergraduate/courses/course/codeC \rightarrow undergraduate/courses/course/titleC))$ |
| 2 | $(univ, (undergraduate/courses/course/codeC \rightarrow undergraduate/@domain))$ |
| 3 | $(univ, (courses/course/codeC \rightarrow courses/course/@domain))$ |
| 4 | $(univ, (courses/course/codeC \rightarrow courses/course/nameC))$ |
| 5 | $(univ, (courses/course/codeC \rightarrow courses/course/@level))$ |
| 6 | $(univ, (\{courses/course/nameC, courses/course/@level\} \rightarrow courses/course/NbHours))$ |

in $X_1$. Notice that the XFD $(univ, (undergraduate/courses/course/codeC \rightarrow undergraduate/courses/course/prerequisitesC))$ in $\mathcal{F}_1$, which states that courses with the same code have the same set of prerequisites, is not in $\mathcal{F}$. The reason is that according to the ontology alignment, this XFD is equivalent to *(univ, (courses/course/codeC → courses/course/prerequisitesC))* in $\mathcal{F}_2$. However, as $\mathcal{F}_2$ does not contain this XFD, documents in $X_2$ may violate it (since the involved concepts exist in $X_2$).

## 4 Preliminaries

Our work uses XFD such as those in [4,6]. An XML document is seen as a tuple $\mathcal{T} = (t, type, value)$. The tree $t$ is the function $t$: $dom(t) \rightarrow \Sigma$ where: (A) $\Sigma = \Sigma_{ele} \cup \Sigma_{att} \cup \{data\}$ is an alphabet; $\Sigma_{ele}$ is the set of element names and $\Sigma_{att}$ is the set of attribute names and (B) $dom(t)$ is the set of positions numbered according to Dewey encoding. Given a tree position $p$, function $type(t,p)$ returns a value in $\{data, element, attribute\}$. Similarly, $value(t,p) = \begin{cases} p & \text{if } type(t,p) = element \\ val \in \mathbf{V} & \text{otherwise} \end{cases}$
where $\mathbf{V}$ is an infinite recursively enumerable domain. □

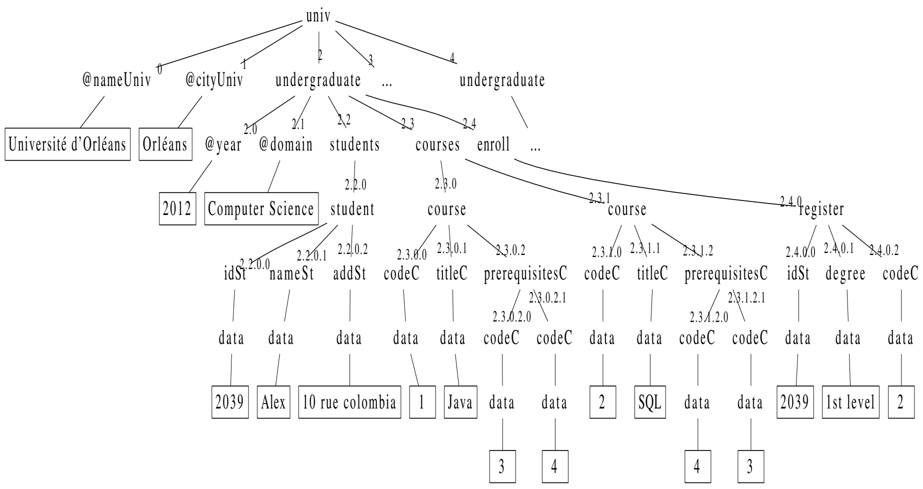

**Fig. 2.** XML document concerning the first degree (undergraduate) at a university

As many other authors, we distinguish two kinds of equality in an XML tree, namely, *value equality and node equality.* Two nodes are *value equal* when they are roots of isomorphic sub-trees. Two nodes are *node equal* when they have the same position number. To combine both equality notions we use the symbol $E$, that can be represented by $V$ for value equality, or $N$ for node equality. Our value equality definition does not take into account the document order. For instance, in Figure 2, nodes in positions 2.3.0.2 and 2.3.1.2 are value equal, but nodes 2.3.0 and 2.3.1 are not.

## 4.1 Linear Paths

Linear paths are used to address parts of an XML document. Let $PL$ **be the language where a path is defined** by $\rho ::= [] \mid l \mid \rho/\rho \mid \rho//l$ where $[]$ is the empty path, $l$ is a label in $\Sigma$, "/" is the concatenation operation, "//" represents a finite sequence (possibly empty) of labels. Notice that $l/[] = []/l = l$ and $[]//l = //l$. We distinguish between paths using the wild-card // and *simple paths* (those with no wild-card) and we denote by $\mathbb{P}$ the set of *all possible rooted simple paths that may occur in an XML tree $t$ respecting a given schema $\mathcal{D}$.*

In this work we consider that the set $\mathbb{P}$ is generated from a given schema $\mathcal{D}$. Notice that $\mathbb{P}$ is a *finite* set of simple (top-down) paths and, in this way, the schema from which it is obtained should ensure a limited *depth* of label repetitions. In other words, the language $L(D)$, obtained from a finite state automaton $D$ (which is built from a given type $\mathcal{D}$), should be finite. Such kind of schema can be expressed, for instance, by a non-recursive DTD. In this way, we are more general than [4,6], where $\mathbb{P}$ was the set containing only all possible paths in *one* given tree.

It is important to notice that one path with wild-card can be associated to a *set* of simple paths in $\mathbb{P}$. This set of simple paths is the language $L(A_P)$, where $A_P$ is a finite-state automaton (FSA) obtained on the basis of the two following steps:

1. From the path language $P$ we construct a finite-state automaton $B_P$ which recognizes the expression $P$ in PL and is similar to restricted regular expressions.
2. $A_P = D \cap B_P$. We retain in $L(A_P)$ the paths which respect the path language $P$ and are simple paths in $\mathbb{P}$.

*Example 1.* We suppose a DTD concerning undergraduate course in a university such that the document of Figure 2 is valid *w.r.t.* it. Let $D$ be the FSA that recognizes $\mathbb{P}$, the language of prefixes of the paths defined by this DTD. Let $P =$ *univ/undergraduate//codeC* and $B_P$ the FSA that recognizes $P$. The set $L(B_P) \cap \mathbb{P}$ contains only the simple paths in $L(B_P)$ that trees respecting $\mathbb{P}$ may have, *i.e.*, { *univ/undergraduate/courses/course/codeC, univ/undergraduate/courses/course/prerequisitesC/codeC, univ/undergraduate/enroll/register/codeC*}. □

A path $P$ is **valid** if: $(i)$ it conforms to the syntax of $PL$, $(ii)$ $L(A_P) \neq \emptyset$, $(iii)$ for all label $l \in P$, if $l = data$ or $l \in \Sigma_{att}$, then $l$ is the last symbol in $P$.

In this work, given a path $P$ in PL we define the following functions:

- *Last*$(P) = l_n$ where $l_n$ is the last label on path $P$.
- *Parent*$(P) = \{l_1/\dots/l_{n-1} \mid l_1/\dots/l_{n-1}/l_n \in L(A_P)$ for $n > 1\}$, the set of simple paths starting at a node labelled by $l_1$ and ending at the parent of $l_n$ (where *Last*$(P) = l_n$).
- A path $Q$ is a prefix of $P$ (we note $Q \preceq_{PL} P$) if $L(A_Q) \subseteq L(PREFIX(A_P))$ where $PREFIX(A_P)$ is the finite state automaton that accepts the language containing all prefixes of $L(A_P)$.
- The **longest common prefix** (or the intersection) of $P$ and $Q$, denoted by $P \cap Q$, describes the set of simple paths $\{P' \cap Q' \mid P' \in L(A_P) \wedge Q' \in L(A_Q)\}$. The longest common prefix of two simple paths $P'$ and $Q'$ (denoted $P' \cap Q'$) is the simple path $R$ where $R \preceq P'$ and $R \preceq Q'$ and there is no path $R'$ such that $R \prec R'$, $R' \preceq P'$ and $R' \preceq Q'$.

*Example 2.* Consider the XML document of Figure 2. The simple path *univ/undergraduate/courses* is a prefix for *univ/undergraduate/courses/course/codeC*. Given $P'$ =*univ/undergraduate/courses/course/codeC* and $Q'$ = *univ/undergraduate/courses/course/prerequisitesC/codeC*, their longest common prefix is *univ/undergraduate/courses/course*.
Given $P$ = *univ//codeC* and $Q$ = *univ//idSt*, the longest common prefix $P \cap Q$ = { *univ/undergraduate, univ/undergraduate/enroll/register*} □

Now, let $I = p_1/\dots/p_n$ be a sequence of positions such that each $p_i$ is a direct descendant of $p_{i-1}$ in $t$. Then $I$ is an **instance of a path** $P$ **over a given tree** $t$ if and only if the sequence $t(p_1)/\dots/t(p_n) \in L(A_P)$. We denote by $Instances(P,t)$ the set of all instances of $P$ over $t$. Functions *Last*, *Parent*, *Prefix* and the *longest common prefix* are extended to path instances in the obvious manner. Notice that the longest common prefix allows the identification of the least common ancestor.

We now remark that, in this paper, we will only deal with **complete trees** (*i.e.*, documents with no missing information). Let $\mathbb{P}$ be a set of simple paths associated to an XML document $\mathcal{T}$. We say that $\mathcal{T}$ is complete *w.r.t.* $\mathbb{P}$ if whenever there exists paths $P$ and $P'$ in the associated $\mathbb{P}$ such that $P' \prec P$ and there exist an instance $I'$ for $P'$ such that node $v'$ is the last node in $I'$, then there exists an instance $I$ for $P$ such that $v$ is the last node in $I$ and $v'$ is an ancestor of $v$. For example, let $\mathbb{P}$ be a set containing paths $R/A/C$, $R/A/D$, $R/B$ and their prefixes. Then, representing trees as terms, we notice that $R(A(C,D),B)$, $R(A(C,D),A(C,D),B)$ are complete trees, while $R(A(C),B)$, $R(A(C,D))$ are no complete trees.

Given two valid paths $P$ and $Q$ over a tree $t$, we want to verify whether two given path instances match on the longest common prefix of $P$ and $Q$. To this end we define **the boolean function** *isInst_lcp*$(P,I,Q,J)$ which returns *true* when all the following conditions hold: (*i*) $I \in Instances(P,t)$; (*ii*) $J \in Instances(Q,t)$ and (*iii*) $I \cap J$ is an instance of a path in $P \cap Q$; otherwise, it returns *false*.

### 4.2 Branching Paths

Now we introduce the notion of branching paths also called a pattern in the literature [5,20]. A branching path is a non-empty set of simple paths having a common prefix. The projection of a tree over a branching path determines the tree positions corresponding to the given path. Thus, as defined below, this projection is a set of prefix closed simple path instances that respect some important conditions.

**Definition 1 (Branching path).** *A branching path is a finite set of prefix-closed (simple) paths on a tree $t$.* □

**Definition 2 (Projection of a tree $\mathcal{T}$ over a branching path $M$).** *Let $M$ be a branching path over a tree $\mathcal{T}$. Let $Long_M$ be the set of paths in $M$ that are not prefix of other paths in $M$. Let SetPathInst be the set of (simple) path instances that verifies:*

1. *For all paths $P \in Long_M$ there is one and only one instance $inst \in Instances(P,t)$ in the set SetPathInst.*
2. *For all $inst \in SetPathInst$ there is a path $P \in Long_M$ such that $inst \in Instances(P,t)$.*
3. *For all instances inst and $inst'$ in SetPathInst, if $inst \in Instances(P,t)$ and $inst' \in Instances(Q,t)$, then $isInst\_lcp(P, inst, Q, inst')$ is true.*

*A projection of $\mathcal{T}$ over $M$, denoted by $\Pi_M(\mathcal{T})$, is a tuple $(t^i, type^i, value^i)$ where $type^i(t^i,p) = type(t,p)$, $value^i(t^i,p) = value(t,p)$ and $t^i$ is a function $\Delta \to \Sigma$ in which:*

- $\Delta = \bigcup_{inst\ \in\ SetPathInst}\{p \mid p \textit{ is a position in inst}\}$
- $t^i(p) = t(p),\ \forall p \in \Delta$ □

Given the projection of two branching paths, $\Pi_{M_1}(\mathcal{T})$ and $\Pi_{M_2}(\mathcal{T})$, the union $\Pi_{M_1}(\mathcal{T}) \cup \Pi_{M_2}(\mathcal{T})$ is naturally obtained by considering all the path instances used to obtain each projection.

*Example 3.* Consider the XML document of Figure 2. Let $M$ be a branching path defined from the set {*univ/undergraduate/courses/course/codeC, univ/undergraduate/courses/course/prerequisitesC*}, *i.e.*, $M$ contains these paths and all their prefixes. An example of a projection of $\mathcal{T}$ over $M$ is the one where $t(\epsilon) = univ$, $t(2) = undergraduate$, $t(2.3) = courses$, $t(2.3.0) = course$, $t(2.3.0.0) = codeC$ and $t(2.3.0.2) = preresquisitesC$. However, if we take $t(\epsilon) = univ$, $t(2) = undergraduate$, $t(2.3) = courses$, $t(2.3.0) = course$, $t(2.3.0.0) = codeC$ and $t(2.3.1.2) = preresquisitesC$, we do not have a projection of $\mathcal{T}$ over $M$. Indeed, in Definition 2, if we consider $P =$ *univ/undergraduate/courses/course/codeC* and its instance $inst = \epsilon/2/2.3/2.3.0/2.3.0.2$ together with $Q =$ *univ/undergraduate/courses/course/prerequisitesC* and its instance $inst' = \epsilon/2/2.3/2.3.1/\text{-}2.3.1.2$ we obtain $isInst\_lcp(P, inst, Q, inst') = false$. Notice that the longest common paths $P \cap Q$ is *univ/undergraduate/courses/course.* □

From Definition 2, we remark that the projection of $\mathcal{T}$ over a branching path $M$ contains exactly one instance of every path in $M$. In the following, when needed, *we denote by* $\varPi_M(\mathcal{T})[P]$ *the unique instance of the simple path* $P$ *in* $\varPi_M(\mathcal{T})$. Indeed, when we write $\varPi_M(\mathcal{T})[P]$ we restrict the projection of $T$ over $M$ to the instance (in the projection) of one simple path $P$.

**Lemma 1.** *Let* $\varPi_M(\mathcal{T})$ *be a projection of a tree* $\mathcal{T}$ *over a branching path* $M$*. For each two simple paths* $P$ *and* $Q$ *in* $M$ *if* $I = \varPi_M(\mathcal{T})[P]$ *and* $J = \varPi_M(\mathcal{T})[Q]$ *then we have* $isInst\_lcp(P, I, Q, J) = true$. $\square$

Now we are interested in building a relation where each tuple corresponds to values determined by a given projection $\varPi_M(\mathcal{T})$.

**Definition 3 (Tuple obtained from Projection).** *Let* $M$ *be a branching path and* $X = \{P_1, \ldots, P_k\}$ *be a set of paths such that* $X \subseteq M$*. Let* $\tau = \varPi_M(\mathcal{T}) = (t^i, type^i, value^i)$ *be a projection of the tree* $\mathcal{T}$ *on* $M$*. Let* $I_j = \varPi_M(\mathcal{T})[P_j]$ *be the only instance of path* $P_j$ *in* $\varPi_M(\mathcal{T})$ *where* $j \in [1, \ldots, k]$*. The tuple corresponding to* $X$ *on* $\tau$*, denoted by* $\tau[X]$*, is defined as*[1]

$$\tau[X] = (P_1 : value^i(t^i, Last(I_1)), \ldots, P_k : value^i(t^i, Last(I_k))).$$

*We denote by* $\tau[P_j]$ *the result of* $P_j : value^i(t^i, Last(I_j))$*. Two tuples* $\tau^1[X]$ *and* $\tau^2[X]$ *are equal w.r.t. the equality list* $E = (E_1, \ldots, E_k)$*, denoted by* $\tau^1[X] =_E \tau^2[X]$*, iff* $\forall j \in [1 \ldots k]$*,* $\tau^1[P_j] =_{E_j} \tau^2[P_j]$. $\square$

The tuple $\tau[X]$ is formed by the values or nodes found in an XML document $\mathcal{T}$ from a projection on branching path $M$, and is constructed by following the named perspective in relational database [1] where the name of attributes in the tuples are known. Notice also that the equality between two tuples may involve different kinds of equality, one for each path.

## 5 Functional Dependencies in XML

Usually, a functional dependency in XML (XFD) is denoted by $X \to Y$ (where $X$ and $Y$ are sets of paths) and it imposes that for each pair of tuples (Definition 3) $t_1$ and $t_2$ if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$. In this paper, our XFD are defined as those in [5,6], generalizing the proposals in [3,17,20,22]. As the dependency can be imposed in a specific part of the document, we specify a *context path.*

**Definition 4 (XML Functional Dependency).** *Given an XML tree* $t$*, an XFD* $f$ *is an expression of the form:*

$$f = (C, (\{P_1\ [E_1], \ldots, P_k\ [E_k]\}\ \to \{Q_1\ [E'_1], \ldots, Q_m\ [E'_m]\}))$$

*where* $C$ *is a path that starts from the root of* $t$ *(context path) ending at the context node;* $\{P_1, \ldots, P_k\}$*,* $\{Q_1, \ldots, Q_m\}$ *are non-empty sets of paths in* $t$*. Both* $P_i$ *(*$i \in [1, \ldots, k]$*) and* $Q_i$ *(*$i \in [1, \ldots, m]$*) start at the context node. The set*

[1] If it is clear by the context, we omit the path when showing a tuple.

*$\{P_1, \ldots, P_k\}$ is the left-hand side (LHS) or determinant of an XFD, and the set $\{Q_1, \ldots, Q_m\}$ is the right-hand side (RHS) or the dependent paths. The symbols $E_1, \ldots, E_k, E'_1, \ldots, E'_m$ represent the equality type associated to each dependency path. When symbols $E_1, \ldots, E_k$ or $E'_1, \ldots, E'_m$ are omitted, value equality is the default choice.* □

Notice that in an $XFD$ the set of paths $\{C/P_1, \ldots, C/P_k, C/Q_1, \ldots, C/Q_m\}$ defines branching paths and that, as in [22], our XFD definition allows the combination of two kinds of equality.

**Definition 5 (XFD Satisfaction).** *Let $\mathcal{T}$ be an XML document and $f = (C, (\{P_1\ [E_1], \ldots, P_k\ [E_k]\} \rightarrow \{Q_1\ [E'_1], \ldots, Q_m\ [E'_m]\}))$ an $XFD$. Let $M$ be a branching path defined from $f$. We say that $\mathcal{T}$ satisfies $f$ (noted by $\mathcal{T} \models f$) if and only if for all $\tau^1 = \Pi^1_M(\mathcal{T})$ and $\tau^2 = \Pi^2_M(\mathcal{T})$ that are projections of $\mathcal{T}$ on $M$ and that coincide at least on their prefix $C$, we have:*
*If $\tau^1[C/P_1, \ldots, C/P_k] =_E \tau^2[C/P_1, \ldots, C/P_k]$ then $\tau^1[C/Q_1, \ldots, C/Q_m] =_{E'} \tau^2[C/Q_1, \ldots, C/Q_m]$ where $E = (E_1, \ldots, E_k)$ and $E' = (E'_1, \ldots, E'_m)$.* □

*Example 4.* Consider the following XFD on the document of Figure 2.

XFD1: $univ//courses, (\{course/codeC\} \rightarrow course/titleC)$
Considering the set of courses of an undergraduate domain, courses having the same code have the same title.
XFD2: $univ, (\{undergraduate//course/codeC\} \rightarrow undergraduate//course/titleC)$. Considering the set of all courses in a university, courses having the same code have the same title.
XFD3: $univ//students, (\{student/idSt\} \rightarrow student[N])$. Considering the set of students of an undergraduate domain, no two students have the same number and each student appears once. □

An XML document $\mathcal{T}$ satisfies a set of XFD $\mathcal{F}$, denoted by $\mathcal{T} \models \mathcal{F}$, if $\mathcal{T} \models f$ for all $f$ in $\mathcal{F}$. Usually it is important to reason whether a given XFD $f$ is also satisfied on $\mathcal{T}$ when $\mathcal{F}$ is satisfied. The following definition introduces this notion.

**Definition 6 (XFD Implication).** *Given a set $\mathcal{F}$ of XFD we say that $\mathcal{F}$ implies $f$, denoted by $\mathcal{F} \models f$, if for every XML tree $\mathcal{T}$ such that $\mathcal{T} \models \mathcal{F}$ then $\mathcal{T} \models f$.* □

Based on the notion of implication we can introduce the definition of closure for a set of XFD.

**Definition 7 (Closure of a set of XFD).** *The closure of a set of XFD $\mathcal{F}$, denoted by $\mathcal{F}^+$, is the set containing all the XFD which are logically implied by $\mathcal{F}$, i.e., $\mathcal{F}^+ = \{f \mid \mathcal{F} \models f\}$.* □

**Notation:** In the rest of this paper, given an XFD $(C, (X \rightarrow A))$ where $X = \{P_1, \ldots, P_n\}$ is a set of paths and $A$ is a path, we use $C/X$ as a shorthand for the set $\{C/P_1, \ldots, C/P_n\}$.

## 6 Axiom System

To find which XFD $f$ are also satisfied when a given set of XFD $\mathcal{F}$ is satisfied we need inference rules that tell how one or more dependencies imply other XFD. In this section we present our axiom system and prove that it is sound (we cannot deduce from $F$ any false XFD) and complete (from a given set $\mathcal{F}$, the rules allow us to deduce all the true dependencies). Our axiom system is close to the one proposed in [21], but has two important differences: our XFD are defined *w.r.t.* a context (and not always *w.r.t.* the root) and we use two kinds of equality.

**Definition 8 (Inference Rules for XFD).** *Given a tree $\mathcal{T}$ and XFD defined over paths in $\mathbb{P}$, our axioms are:*

*A1:* ***Reflexivity*** $(C, (\{P_1\,[E_1], \ldots, P_n\,[E_n]\} \rightarrow P_i\,[E_i])), \forall i \in [1 \ldots n]$.

*A2:* ***Augmentation*** *If* $(C, (\{P_1\,[E_1], \ldots, P_n\,[E_n]\} \rightarrow \{Q_1\,[E'_1], \ldots, Q_m\,[E'_m]\}))$ *then* $(C, (\{R\,[E_r], P_1\,[E_1], \ldots, P_n\,[E_n]\} \rightarrow \{R\,[E_r], Q_1\,[E'_1], \ldots, Q_m\,[E'_m]\}))$.

*A3:* ***Transitivity*** *If* $(C, (\{P_1\,[E_1], \ldots, P_n\,[E_n]\} \rightarrow \{Q_1\,[E'_1], \ldots, Q_m\,[E'_m]\}))$ *and* $(C, (\{Q_1\,[E'_1], \ldots, Q_m\,[E'_m]\} \rightarrow S\,[E_s]))$ *then* $(C, (\{P_1\,[E_1], \ldots, P_n\,[E_n]\} \rightarrow S\,[E_s]))$.

*A4:* ***Branch Prefixing*** *If* $(C, (\{P'_1\,[E'_1], \ldots, P'_n\,[E'_n]\} \rightarrow P_{n+1}\,[E_{n+1}]))$ *and there exist paths* $C/P_1, \ldots, C/P_n$ *(not necessarily distinct) such that*
*(i)* $P'_i \cap P_{n+1} \preceq_{PL} P_i$ *and*
*(ii)* $P_i \preceq_{PL} P'_i$ *or* $P_i \preceq_{PL} P_{n+1}$
*then* $(C, (\{P_1\,[E_1], \ldots, P_n\,[E_n]\} \rightarrow P_{n+1}\,[E_{n+1}]))$.

*A5:* ***Ascendency*** *If* $Q$ *is a prefix for* $P$ *then* $(C, (P\,[N] \rightarrow Q\,[N]))$.

*A6:* ***Attribute Uniqueness*** *If* $Last(P) \in \Sigma_{att}$ *then* $(C, (Parent(P)\,[E] \rightarrow P\,[E]))$.

*A7:* ***Root Uniqueness*** $(C, (\{P_1\,[E_1], \ldots, P_n\,[E_n]\} \rightarrow []\,[E_{n+1}]))$.

*A8:* ***Context Path Extension*** *If* $(C, (\{P_1\,[E_1], \ldots, P_n\,[E_n]\} \rightarrow P_{n+1}\,[E_{n+1}]))$ *and there is a path* $Q$ *such that* $P_1 = Q/P'_1, \ldots, P_{n+1} = Q/P'_{n+1}$ *then* $(C/Q, (\{P'_1\,[E_1], \ldots, P'_n\,[E_n]\} \rightarrow P'_{n+1}\,[E_{n+1}]))$.

*A9:* ***Node Equality to Value Equality*** $\forall P, (C, (P\,[N] \rightarrow P\,[V]))$.

*Example 5.* A given university has one or more undergraduate specialities (first degree) and, for each of them, we store its domain and year together with information concerning students, courses and enrolment. Figure 2 shows a part of this XML document over which we illustrate the intuitive meaning of axioms A4-A9. The intuition of the three first axioms (A1-A3) is the same as in relational.

A4: If $(univ, (\{undergraduate/@domain, courses//codeC\} \rightarrow undergraduate/enroll//degree))$ then we can say that: $(univ, (\{undergraduate/@domain, undergraduate/courses\} \rightarrow undergraduate/enroll//degree))$ or $(univ, (\{undergraduate/@domain, undergraduate/courses/course\} \rightarrow undergraduate/enroll//degree))$ or $(univ, (undergraduate \rightarrow undergraduate/enroll//degree))$.

The initial XFD states that all courses having *codeC* in the same domain correspond to the same degree. From this XFD, we can deduce, among others, the XFD $(univ, (undergraduate \rightarrow undergraduate/enroll//degree))$ stating that an undergraduate speciality is associated to only one degree (*e.g.*, Bachelor's).

A5: Given a path $P = undergraduate//register/idSt$, we can derive $(univ, (\{undergraduate//register/idSt\} \rightarrow undergraduate//register))$.

A6: Given $P = undergraduate/@year$, we derive that $(univ, (\{undergraduate[N]\} \rightarrow undergraduate/\ @year[N]))$.

A8: If $(univ/undergraduate, (students/student/idSt \rightarrow students/student/nameSt))$ then $(univ/undergraduate/students, (student/idSt \rightarrow student/nameSt))$. If, in the context of an undergraduate domain, the *idSt* identifies the name of a student; this is also true in the context of *students*.

A9: When we have a node equality, for instance, for *univ/undergraduate//course*, it means that we are considering a specific, uniquely referred, course in our document. Thus, $(univ, (\{undergraduate//course[N]\} \rightarrow undergraduate//course[V]))$ is a valid XFD.

Notice that $A5$ does not hold when dealing with value equality. The tree on Figure 2 violates the XFD $(univ, (\{undergraduate//course/prerequisitesC\,[V]\} \rightarrow undergraduate//course\,[V]))$. Indeed $Last(2.3.0.2) =_V Last(2.3.1.2)$ but $Last(2.3.0) \neq_V Last(2.3.1)$.

Remark that although we have value equality, the following rule $(C, (P[V] \rightarrow P/Q\,[V]))$ does not hold. Let us consider the XFD $(univ, (\{undergraduate//prerequisitesC\,[V]\} \rightarrow undergraduate//prerequisitesC/codeC\,[V]))$. The tree on Figure 2 does not satisfy this XFD because we have $Last(2.3.0.2) =_V Last(2.3.1.2)$ but $Last(2.3.0.2.0) \neq_V Last(2.3.1.2.0)$. □

The set of axioms in Definition 8 establishes an inference system with which one can derive other XFD.

**Definition 9 (XFD Derivation).** *Given a set $\mathcal{F}$ of XFD, we say that an XFD $f$ is derivable from the functional dependencies in $\mathcal{F}$ by the set of inference rules in Definition 8, denoted by $\mathcal{F} \vdash f$, if and only if there is a sequence of XFD $f_1, f_2, \ldots, f_n$ such that (i) $f = f_n$ and (ii) for all $i = 1, \ldots, n$ the XFD $f_i$ is in $\mathcal{F}$ or it is obtainable from $f_1, f_2, \ldots f_{i-1}$ by means of applying an axiom A1-A9 (from Definition 8).* □

Our axiom system is sound and complete. The proofs are summarized in Appendix A and B and in [2] one can find more detailed versions. Additional inferences rules (Union, Decomposition, Pseudotransitivity and Subtree Uniqueness) can be derived from axioms of Definition 8 as we show in [2]. Notice that as we have the Union and Decomposition axioms, an important consequence is that an XFD $(C, (\{P_1\ [E_1], \ldots, P_k\ [E_k]\}\ \rightarrow \{Q_1\ [E'_1], \ldots, Q_m\ [E'_m]\}))$ holds if and only if $(C, (\{P_1\ [E_1], \ldots, P_k\ [E_k]\}\ \rightarrow \{Q_i\ [E'_i]\}))$ holds for $i \in [1, \ldots, m]$. Thus, having a single path on the right-hand side of an XFD is sufficient. Once we have our axiom system, we can define the closure of a set of paths *w.r.t.* a set of XFD.

**Definition 10 (Closure of a set of Paths).** *Let $X$ be a set of paths and let $C$ be a path defining a context. Let $E = (E_1, \ldots, E_n)$ be the equality list associated to $X$. The closure of $(C, X)$ with respect to $\mathcal{F}$, denoted by $(C, X[E])^+_{\mathcal{F}}$, is the set of paths $\{C/P_1[E'_1], \ldots, C/P_m[E'_m]\}$ such that $(C, (X[E] \rightarrow \{P_1[E'_1], \ldots, P_m[E'_m]\}))$ can be deduced from $\mathcal{F}$ by the axiom system in Definition 8. In other words, $(C, X[E])^+_{\mathcal{F}} = \{C/P[E'] \mid \mathcal{F} \vdash (C, (X[E] \rightarrow P[E']))\}$. When there is no ambiguity about the set $\mathcal{F}$ being used, we just note $(C, X[E])^+$.* □

To compute $(C, X[E])^+$ we start with a set $T$ containing all the prefixes of the paths in $X$. Then we build a set $V$ containing all the paths ending on attributes and having a path in $T$ as its parent. Starting with $X^{(0)} = T \cup V$ we compute each $X^{(i+1)}$ from $X^{(i)}$ by applying the axiom system on $\mathcal{F}$. At each step, new sets $T$ and $V$ are computed and added to $X^{(i)}$. The loop ends when no new path can be added to $X^{(i)}$. In [2] we present this algorithm together with the proof of its soundness and completeness.

We also define two other functions, namely *closure1Step* and *inverseClosure1Step*. Function *closure*1*Step* computes one step of the closure of a set of paths. Its implementation consists in applying the same algorithm used to find $(C, X[E])^+$, in order to compute $X^{(1)}$. The result is a set of paths. Function *inverseClosure*1*Step* considers XFD inversely and computes an "inverse closure" one step backward. For instance, given a set of paths $X$ the function finds all sets of paths $Z$ for which we have $C/Z \rightarrow C/X$. The computed result is a set $S$ containing sets of paths.

These functions are going to be used in the following section in order to compute *cover*$\mathcal{F}$.

## 7 Computing Functional Dependencies for Interoperability

### 7.1 Algorithm for Computing *cover*$\mathcal{F}$

Given XFD sets $\mathcal{F}_1, \ldots, \mathcal{F}_n$, let $\mathcal{F} = (\mathcal{F}_1^+ \cap \cdots \cap \mathcal{F}_n^+) \cup (K_1 \cup \cdots \cup K_n)$ where for $1 \leq i \leq n$, $\mathcal{F}_i^+$ is the closure of $\mathcal{F}_i$ and $K_i$ is a set of XFD containing all XFD $f$ which can be obtained from $\mathcal{F}_i$ but that cannot be violated by documents in $X_j$ (for $j \neq i$). In [2] we have proved that $\mathcal{F}$ is the biggest set of XFD that are not in contradiction with any set $\mathcal{F}_1, \ldots, \mathcal{F}_n$. In other words, all documents in $X_1, \ldots, X_n$ valid *w.r.t.* $\mathcal{F}_1, \ldots, \mathcal{F}_n$ should stay valid *w.r.t.* $\mathcal{F}$. Our goal is to propose an algorithm that computes a set *cover*$\mathcal{F}$ which is equivalent to $\mathcal{F}$ (*cover*$\mathcal{F} \equiv \mathcal{F}$), and usually the number of XFD in *cover*$\mathcal{F}$ is much smaller than the number of XFD in $\mathcal{F}$.

Algorithm 1 generates *cover*$\mathcal{F}$ as expected. As input, the algorithm receives the local sets of XFD together with the set of possible paths given by each local schema. Notice that for the sake of simplicity, we suppose only two local sources, but Algorithm 1 can be easily extended for $n$ local sources. Translation functions $\Phi_1$ and $\Phi_2$ are available. These functions work on the translation table (obtained from the ontology alignment $\mathcal{A}$): given a path $P$ from, for instance $\mathbb{P}_2$, $\Phi_1(P)$ gives its equivalent path in $\mathbb{P}_1$, if it exists; otherwise it returns the identity. The function $\Phi_2$ works on a symmetric way. Indeed, we note $i$ and $\bar{i}$ to indicate symmetric sources (*e.g.*, when $i = 1$, $\bar{i} = 2$).

Algorithm 1 considers each local set $\mathcal{F}_i$. Then, each XFD $f = (C, (X \rightarrow B))$ in $\mathcal{F}_i$ is checked and added to $cover\mathcal{F}$ when one of the following properties holds:
(i) There is no path in the source $\bar{i}$ equivalent to the right-hand side of $f$ (line 5). Thus, documents in $\bar{i}$ do not violate $f$.
(ii) There is no set of paths in the source $\bar{i}$ equivalent to the set on the left-hand side of $f$ (line 7). Since no set of paths in the source $\bar{i}$ correspond to $X$, no document in $\bar{i}$ violates $f$.
(iii) In the source $\bar{i}$, there is a path equivalent to $C/B$ that belongs to the closure of a set of paths equivalent to $C/X$ (line 9). Therefore, XFD $f$ exists in both sources and can be added to $cover\mathcal{F}$.

From line 12 to 18, Algorithm 1 takes the fact into account that working with $\mathcal{F}_i$, some XFD in $\mathcal{F}_i^+$ may be neglected. To understand this problem, let us consider sets $\mathcal{F}_1$ and $\mathcal{F}_2$ from which we can derive an XFD $f = (C, (X \rightarrow B))$ by different derivation sequences. Suppose that in $\mathcal{F}_1$ we have $f_1, \ldots, f_k, \ldots, f$ while in $\mathcal{F}_2$ we have $f'_1, \ldots, f'_k, \ldots, f$. Moreover, we assume that, due to conditions stated in lines 5, 7 and 9, the dependencies $f_k$ and $f'_k$ are not included in $\mathcal{F}$ and, thus, the derivation of $f$ is not possible from the new set $cover\mathcal{F}$ built by Algorithm 1. This would be a mistake, since $f$ is derived by both $\mathcal{F}_1$ and $\mathcal{F}_2$. One solution would be to start with (in line 4) the closure of $\mathcal{F}_1$ and $\mathcal{F}_2$. However, this solution implies the generation of a too big and, thus, not manipulable set of XFD. Algorithm 1 does better: when the test in line 9 fails, it computes all XFD $f_j = (C, (Y \rightarrow A))$ such that:

$(i)$ $C/X \in (C, Y)^+$ and $A = B$ or
$(ii)$ $C/Y = C/X$ and $C/A \in (C, B)^+$ or
$(iii)$ $C/A = C/B$ and $f_j$ is obtained by using Axiom $A4$ on $f$ or
$(iv)$ $Y = X \cup Y_1$ and $f_j$ is obtained by using Axiom $A2$ on $f$ to obtain $f' = (C, (X, Y_1 \rightarrow B, Y_1))$, and then using Axiom $A3$ on $f'$ and $f'' = (C, (B, Y_1 \rightarrow A)) \in G$.

Tests from lines 12-18 are then performed on these computed XFD. In this way, we do not compute the entire closure of a set $\mathcal{F}_i$ but, when necessary, we calculate a part of it. This computation is done by using $closure1Step$ and $inverseClosure1Step$. The following example illustrates the computation performed in lines 12-18 of Algorithm 1.

*Example 6.* Let $\mathcal{F}_1 = \{(C, (A \rightarrow B)), (C, (B \rightarrow M)), (C, (M \rightarrow D)), (C, (D \rightarrow E)), (C, (O \rightarrow Z))\}$ and let $\mathcal{F}_2 = \{(C, (A \rightarrow B)), (C, (B \rightarrow M)), (C, (B \rightarrow O)), (C, (O \rightarrow E)), (C, (D \rightarrow N))\}$. Without lines 12-18 in Algorithm 1, the XFD $(C, (A \rightarrow E))$, derivable from both $\mathcal{F}_1$ and $\mathcal{F}_2$, would not be derived from $cover\mathcal{F}$.
Let us consider part of the execution of Algorithm 1. Table 4 shows the XFD we obtain when considering each XFD in $\mathcal{F}_1$ (line 3 of Algorithm 1). The first

**Algorithm 1.** Computation of $cover\mathcal{F}$ (set of XFD ensuring the interoperability of $S$ *w.r.t.* $S_1$ and $S_2$)

**Input:**
- A set of XFD $\mathcal{F}_1$ for schema $\mathcal{D}_1$
- A set of XFD $\mathcal{F}_2$ for schema $\mathcal{D}_2$
- The set of paths $\mathbb{P}_1, \mathbb{P}_2$ specified by $\mathcal{D}_1$ and $\mathcal{D}_2$
- Translation functions $\Phi_1$ and $\Phi_2$

**Output:** The set of XFD $cover\mathcal{F}$ for the integrated system

```
1: $cover\mathcal{F} = \emptyset$
2: for $i = 1$ to 2 do
3:     $G = \mathcal{F}_i$
4:     for each $(C, (X \to B)) \in G$ do
5:         if $\Phi_{\bar{i}}(C/B) \notin \mathbb{P}_{\bar{i}}$ then
6:             $cover\mathcal{F} = cover\mathcal{F} \cup \{(C, (X \to B))\}$
7:         else if $\Phi_{\bar{i}}(C/X) \not\subseteq \mathbb{P}_{\bar{i}}$ then
8:             $cover\mathcal{F} = cover\mathcal{F} \cup \{(C, (X \to B))\}$
9:         else if $\Phi_{\bar{i}}(C/B) \in \Phi_{\bar{i}}(C, X)^+_{\mathcal{F}_{\bar{i}}}$ then
10:            $cover\mathcal{F} = cover\mathcal{F} \cup \{(C, (X \to B))\}$
11:        else
12:            $H = closure1Step(C, B, \mathcal{F}_i) \setminus \{C/B\}$
13:            $G = G \cup \{(C, (X \to D)) \mid C/D \in H\}$
14:            $K = inverseClosure1Step(C, X, \mathcal{F}_i) \setminus \{C/X\}$
15:            $G = G \cup \{(C, (Y \to B)) \mid C/Y \in K\}$
               % Recall that $C/Y$ is a shorthand for $\{C/A_1, \ldots, C/A_n\}$ and that $K$ is
               a set of paths sets.
16:            $G = G \cup \{(C, (Z \to B)) \mid (C, (Z \to B))$ is obtained by using Axiom $A4$
               on $(C, (X \to B))\}$
17:            % Notice that $Z$ is a set of prefixes of paths in $X$ or $B$
18:            $G = G \cup \{(C, (X, W \to V)) \mid (C, (X, W \to V))$ is obtained by using
               the Axioms $A2$, $A3$ on $(C, (X \to B))$ and $(C, (B, W \to V))$ where
               $(C, (B, W \to V)) \in G\}$
19:        end if
20:    end for
21: end for
22: return $cover\mathcal{F}$
```

column of this table shows the XFD in $G$ being verified. The second column indicates XFD that are added to $G$ due to lines 12-18. Finally the last column shows XFD that are inserted in $cover\mathcal{F}$.

Table 4 is obtained by following the execution of Algorithm 1. For instance, let us consider the third line in Table 4: the case when the XFD $(C, (M \to D))$ in $\mathcal{F}_1$ is taken in line 4 of Algorithm 1. This XFD does not verify any condition among conditions in lines 5, 7 and 9. When line 12 is executed, the set $H = \{C/E\}$ is computed, since $closure1Step(C,\ D,\ \mathcal{F}_1)$ gives $\{C/D, C/E\}$. Thus, the XFD $(C, (M \to E))$ is added to $G$ (line 13). When line 14 is executed,

**Table 4.** Computation of (part) of *cover*$\mathcal{F}$: XFD obtained when considering $\mathcal{F}_1$

| $G$ **(XFD being considered)** | **Add to** $G$ | **XFD added to** *cover*$\mathcal{F}$ |
|---|---|---|
| $(C,(A \to B))$ | | $(C,(A \to B))$ (cond. line 9) |
| $(C,(B \to M))$ | | $(C,(B \to M))$ (cond. line 9) |
| $(C,(M \to D))$ | $(C,(M \to E))$ $(C,(B \to D))$ $(C,([] \to D))$ | |
| $(C,(D \to E))$ | $(C,(M \to E))$ $(C,([] \to E))$ | |
| $(C,(O \to Z))$ | | $(C,(O \to Z))$ (cond. line 5) |
| $(C,(M \to E))$ | $(C,(B \to E))$ $(C,([] \to E))$ | |
| $(C,(B \to D))$ | $(C,(B \to E))$ $(C,(A \to D))$ $(C,([] \to D))$ | |
| $(C,(B \to E))$ | | $(C,(B \to E))$ (cond. line 9) |
| $(C,(A \to D))$ | $(C,(A \to E))$ $(C,([] \to D))$ | |
| $(C,(A \to E))$ | | $(C,(A \to E))$ (cond. line 9) |

the set $K = \{\{C/B\}\}$ is computed, since *inverseClosure1Step*$(C, M, \mathcal{F}_1)$ gives $\{\{C/B\}, \{C/M\}\}$. Thus, the XFD $(C,(B \to D))$ is added to $G$ (line 15). When line 16 is executed, the XFD $(C,([] \to D))$ is added to $G$. Notice that these three XFD are analysed later (lines 6 and 7 of Table 4). They are not included in *cover*$\mathcal{F}$, but generate other XFD as, for instance, $(C,(A \to E))$, which is finally added to *cover*$\mathcal{F}$. □

## 7.2 Properties of *cover*$\mathcal{F}$

In this section we prove that Algorithm 1 works correctly, and fulfills our goals. First we introduce Lemma 2, telling us which XFD should be added to the set $\mathcal{F} \setminus \{f\}$ in order to ensure the derivation of $\mathcal{F}^+$, except for $f$. Indeed, the derivation of $f$ from the new set $G$ is neither guaranteed nor proscribed.

**Lemma 2.** Let $\mathcal{F}$ be a set of XFD such that $(C,(X \to Y)) \in \mathcal{F}$. Let $(C,(Z_1 \to Z_2))$ be an XFD different from $(C,(X \to Y))$. If $\mathcal{F} \vdash (C,(Z_1 \to Z_2))$ then $G \vdash (C,(Z_1 \to Z_2))$ where $G$ is obtained from $\mathcal{F}$ as follows:

$$G = \mathcal{F} \cup \mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3 \cup \mathcal{F}_4 \setminus \{(C,(X \to Y))\}$$

where $\mathcal{F}_1 = \{(C,(X \to V)) \mid V \in closure1Step(C, Y, \mathcal{F})\}$,
$\mathcal{F}_2 = \{(C,(W \to Y)) \mid W \in inverseClosure1Step(C, X, \mathcal{F})\}$,
$\mathcal{F}_3 = \{(C,(\{P'_1, \ldots, P'_n\} \to Y)) \mid \{P'_1, \ldots, P'_n\}$ respects conditions for applying Axiom A4 on $(C,(X \to Y))\}$,
$\mathcal{F}_4 = \{(C,(X, W \to V)) \mid (C,(Y, W \to V)) \in \mathcal{F}\}$. □

*Sketch of proof:* Since $\mathcal{F} \vdash (C, (Z_1 \rightarrow Z_2))$, there exists a sequence $\alpha$ of XFD containing XFD in $\mathcal{F}$ such that $\alpha$ derives $(C, (Z_1 \rightarrow Z_2))$. The crucial point of the proof is when we suppose that $\alpha$ contains $(C, (X \rightarrow Y))$. Thus, $\alpha$ has sub-sequences for which one of the following conditions holds:

– it derives the set of paths $X$ in one step or
– it derives the paths $Y_1, \ldots, Y_n \in \mathit{closure1Step}(C, Y, \mathcal{F})$ or
– it derives path $Y$ in one step by using Axiom A4 on $(C, (X \rightarrow Y))$ or
– it derives path $V$ from $X, W$ where $(C, (Y, W \rightarrow V)) \in \mathcal{F}$.

The proof consists in replacing the XFD $(C, (X \rightarrow Y))$ and all sub-sequences of $\alpha$ respecting the above conditions, by some of the new XFD which are added to $\mathcal{F}$ for obtaining $G$. By considering $G$ and the new derivation sequence (obtained after replacing XFD in $\alpha$) we can derive $(C, (Z_1 \rightarrow Z_2))$. □

Now, given two sets of XFD, $\mathcal{F}_i$ and $\mathcal{F}_{\bar{i}}$, we define set $\mathcal{K}_i$ of XFD which contains all the XFD $f$ which can be obtained from $\mathcal{F}_i$ but that cannot be violated by documents in $X_{\bar{i}}$ due to one of the two reasons:
(a) the right-hand side of $f$ is a path $B$ which belongs to $\mathbb{P}_i$ but not to $\mathbb{P}_{\bar{i}}$ or
(b) the left-hand side of $f$ is a set of paths $X$ which is included in $\mathbb{P}_i$ but not in $\mathbb{P}_{\bar{i}}$.

Formally, we have $\mathcal{K}_i = \{X \rightarrow A \mid X \rightarrow A \in \mathcal{F}_i^+$ and $[((X \subseteq \mathbb{P}_i)$ and $(X \nsubseteq \mathbb{P}_{\bar{i}}))$ or $(A \in (\mathbb{P}_i \setminus \mathbb{P}_{\bar{i}})]\}$.

In [2] we show an algorithm, starting with $\mathcal{F}_1^+$ and $\mathcal{F}_2^+$, instead of $\mathcal{F}_1$ and $\mathcal{F}_2$, that computes the set $\mathcal{F} = (\mathcal{F}_1^+ \cap \mathcal{F}_2^+) \cup \mathcal{K}_1 \cup \mathcal{K}_2$. We prove some properties of $\mathcal{F}$. This set $\mathcal{F}$ is the biggest set of XFD that does not violate any document in $X_i$ and $X_{\bar{i}}$.

**Theorem 1.** *The set cover$\mathcal{F}$, returned by Algorithm 1, is equivalent to (or is a cover of) the set of XFD $\mathcal{F} = (\mathcal{F}_1^+ \cap \mathcal{F}_2^+) \cup \mathcal{K}_1 \cup \mathcal{K}_2$ (cover$\mathcal{F} \equiv \mathcal{F}$).* □

*Sketch of proof:* For proving that $\mathit{cover}\mathcal{F} \equiv \mathcal{F}$, we will prove that: $(A1)$ $\forall f \in \mathit{cover}\mathcal{F}$, $\mathcal{F} \vdash f$ and $(A2)$ $\forall f \in \mathcal{F}$, $\mathit{cover}\mathcal{F} \vdash f$.

$(A1)$ By the following Algorithm 1, we can easily prove that each XFD added to $\mathit{cover}\mathcal{F}$ is also in $\mathcal{F}$. Thus, we have $\mathit{cover}\mathcal{F} \subseteq \mathcal{F}$ which is stronger than just proving that $\mathcal{F} \vdash f$ for any XFD $f \in \mathit{cover}\mathcal{F}$.

$(A2)$ Let $f = (C, (Y \rightarrow A))$ be an XFD in $\mathcal{F}_1^+ \cap \mathcal{F}_2^+$. Thus, we know that $C/A \in (C, Y)^+_{\mathcal{F}_1}$ and $C/A \in (C, Y)^+_{\mathcal{F}_2}$. Since $C/A \in (C, Y)^+_{\mathcal{F}_1}$, there is a derivation sequence $\alpha = f_1, \ldots, f_n$ which derives $f$. If $\mathit{cover}\mathcal{F}$ contains all the XFD of $\mathcal{F}_1$ taking part in $\alpha$ then we have $\mathit{cover}\mathcal{F} \vdash f$. Otherwise there is at least one XFD of $\mathcal{F}_1$ (denote it by $f_k$) that takes part in $\alpha$ but does not belong to $\mathit{cover}\mathcal{F}$. Since $f_k \notin \mathit{cover}\mathcal{F}$ then from lines 12-16, we know that $f_k$ is deleted from $G$ and that some other XFD $h$ is inserted in $G$. By using Lemma 2, we have $G \vdash f$. If all new functional dependencies $h$ satisfy conditions in lines 5, 7, 9 they are added to $\mathit{cover}\mathcal{F}$. Otherwise, they are analysed in lines 12-16 and the process goes on until $f$ is added to $G$ and, thus, to $cover\mathcal{F}$. With the same arguments we can prove that $f \in cover\mathcal{F}^+$ when $f \in \mathcal{K}_1$ or $f \in \mathcal{K}_2$. $\square$

### 7.3 Complexity of Our Method

Algorithm 1 depends on the algorithm that computes the closure of a set of paths ($(C, X[E])^+$), and on the algorithm that computes just one step of the inverse closure of a set of paths.

The running time of the closure algorithm, in the worst (unlikely) case, is $O(|\mathbb{P}|^2 \cdot (|f| \cdot |\mathcal{F}| + |\mathbb{P}|))$ where $|\mathcal{F}|$ is the cardinality of $\mathcal{F}$ and $|f|$ is the size of the longest XFD in $\mathcal{F}$. The running time of the $inverseClosure1Step$ algorithm is $O((|f|^n \cdot |\mathcal{F}|)^{|X|})$ where $|f|$ is the size of the longest XFD in $\mathcal{F}$, $n$ is the number of paths on the left-hand side of $f$ and $|X|$ is the cardinality of the set of paths $X$ on which the function $inverseClosure1Step$ is performed.

In the worst case, Algorithm 1 will treat about $|\mathcal{F}_i| \cdot |\mathbb{P}_i|$ functional dependencies for each set $\mathcal{F}_i$. The worst case occurs when for each XFD $f = (C, (X \rightarrow P))$ in $\mathcal{F}_i$, $(C, X)^+$ contains $|\mathbb{P}_i|$ paths and just one path is added to $(C, X)^+$ in each step of the loop of the closure algorithm and no XFD is added to $cover\mathcal{F}$. Hence, in this case, lines 12-18 of Algorithm 1 will be executed $|\mathbb{P}_i|$ times for each XFD in $\mathcal{F}_i$. The complexity of Algorithm 1 is $O(|\mathcal{F}_i| \cdot |\mathbb{P}_i| \cdot (g + h))$ where $g$ is the complexity of the closure algorithm and $h$ is the complexity of the $inverseClosure1Step$ algorithm. The variables that are determinants in the complexity are the cardinality of $\mathcal{F}$ and $\mathbb{P}$. In practice $|X|$ and $n$ are not greater than 5 and, thus, have little importance when compared with the size of $\mathcal{F}$ and $\mathbb{P}$.

## 8 Experimental Results

In order to examine the performance of Algorithm 1, we run several experiments on synthetic data. Algorithm 1 has as input two local systems $S_1 = (\mathcal{D}_1, \mathcal{F}_1, O)$ and $S_2 = (\mathcal{D}_2, \mathcal{F}_2, O)$, and computes the set $cover\mathcal{F}$ which contains only the XFD for which no violation is possible when considering document sets from $S_1$ and $S_2$. Recall that we assume the existence of a pre-processing step where the correspondence among paths on the different local documents is established. This pre-processing step is built on the basis of an ontology alignment but it is out of purpose in this paper. In this section we assume that this correspondence has already been done: paths are represented on the basis of a common ontology $O$.

We take into account two parameters in the experiments: ($i$) the number of paths obtained from $\mathcal{D}_1$ and $\mathcal{D}_2$, and ($ii$) the number of XFD in $| \mathcal{F}_1 | + | \mathcal{F}_2 |$.

Tree $\mathcal{T}$ (Figure 3) guides the way we perform our experiments. $\mathcal{T}$ is built by repeating the pattern tree in Figure 3 several times. To perceive the difference between the sub-trees of $\mathcal{T}$, we relabel the nodes of the pattern tree by adding the index $k$ ($k \geq 1$). We say *sub-tree* $k$ to refer to the $k$th tree pattern in $\mathcal{T}$. For example, in Figure 3, $A_{1,1}$ refers to element $A_1$ of subtree $k = 1$ while and $A_{1,2}$ refers to element $A_1$ of subtree $k = 2$.

Our experiments consist in generating *cover*$\mathcal{F}$ from sets $\mathcal{F}_1$ and $\mathcal{F}_2$ which increase at each test by assuming the existence of bigger sets of paths $\mathbb{P}_1$ and $\mathbb{P}_2$ and, therefore, larger trees $\mathcal{T}$. In the text, we usually refer to tree $\mathcal{T}$ to indicate the type of documents (the schema) we are dealing with. In this context, let us define $\mathbb{P}_1^j$ as the set of paths containing all the paths in the tree $\mathcal{T}$ except the paths $C/R_{1,k}/G_{1,k}$ (with $k \leq j$), and $\mathbb{P}_2^j$ as the set of paths containing all the paths in the tree $\mathcal{T}$ except the paths $C/R_{1,k}/F_{1,k}$ (with $k \leq j$). We suppose that the set of paths $\mathbb{P}_1^j$ (respectively $\mathbb{P}_2^j$) is generated from $\mathcal{D}_1$ (respectively $\mathcal{D}_2$).

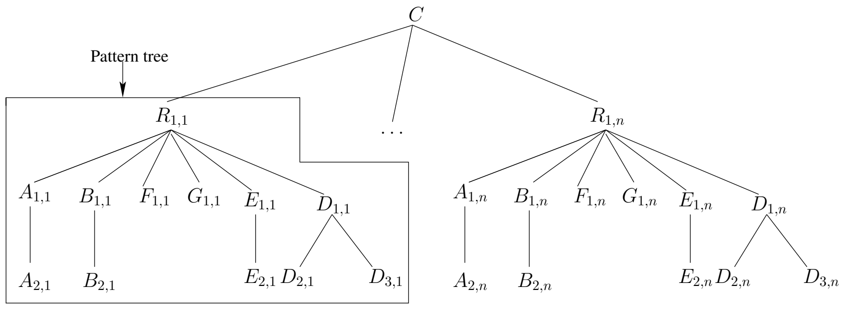
C
Pattern tree
$R_{1,1}$
...
$R_{1,n}$
$A_{1,1}$ $B_{1,1}$ $F_{1,1}$ $G_{1,1}$ $E_{1,1}$ $D_{1,1}$
$A_{2,1}$ $B_{2,1}$ $E_{2,1}$ $D_{2,1}$ $D_{3,1}$
$A_{1,n}$ $B_{1,n}$ $F_{1,n}$ $G_{1,n}$ $E_{1,n}$ $D_{1,n}$
$A_{2,n}$ $B_{2,n}$ $E_{2,n}$ $D_{2,n}$ $D_{3,n}$

**Fig. 3.** Tree $\mathcal{T}$ built by repeating $n$ times the pattern tree

The set of XFD $\mathcal{F}_1^j$ (respectively $\mathcal{F}_2^j$) is defined over paths in $\mathbb{P}_1^j$ (respectively $\mathbb{P}_2^j$). Table 5 shows the XFD in $\mathcal{F}_1^j$ and $\mathcal{F}_2^j$. Sets $\mathcal{F}_1^j$ and $\mathcal{F}_2^j$ contain both XFD (1) and (2). However, XFD (3*a*), (4*a*) and (5*a*) are only in $\mathcal{F}_1^j$ and XFD (3*b*), (4*b*) and (5*b*) are only in $\mathcal{F}_2^j$. With XFD (4*a*) and (5*a*), we can derive the XFD (6) $(C/R_{1,k}, (\{A_{1,k}/A_{2,k}, B_{1,k}/B_{2,k}\} \rightarrow E_{1,k}/E_{2,k}))$ and with XFD (4*b*) and (5*b*), we can also derive the XFD (6). Hence, $\mathcal{F}_1^j$ and $\mathcal{F}_2^j$ derive XFD (6) but by different ways. We can remark that $|\mathcal{F}_1^1| = |\mathcal{F}_2^1| = 5$, $|\mathcal{F}_1^2| = |\mathcal{F}_2^2| = 10$ and $\mathcal{F}_1^1 \subset \mathcal{F}_1^2$, $\mathcal{F}_2^1 \subset \mathcal{F}_2^2$.

**Table 5.** Contents of the XFD sets $\mathcal{F}_1^j$ and $\mathcal{F}_2^j$ used in the experiments

| $\mathcal{F}_1^j$ | $\mathcal{F}_2^j$ |
|---|---|
| (1) $(C/R_{1,k}, (\{A_{1,k}, B_{1,k}\} \rightarrow D_{1,k}))$ | (1) $(C/R_{1,k}, (\{A_{1,k}, B_{1,k}\} \rightarrow D_{1,k}))$ |
| (2) $(C/R_{1,k}, (\{D_{1,k}\} \rightarrow E_{1,k}))$ | (2) $(C/R_{1,k}, (\{D_{1,k}\} \rightarrow E_{1,k}))$ |
| (3*a*) $(C/R_{1,k}, (\{E_{1,k}\} \rightarrow F_{1,k}))$ | (3*b*) $(C/R_{1,k}, (\{E_{1,k}\} \rightarrow G_{1,k}))$ |
| (4*a*) $(C/R_{1,k}, (\{A_{1,k}/A_{2,k}, B_{1,k}/B_{2,k}\} \rightarrow D_{1,k}/D_{2,k}))$ | (4*b*) $(C/R_{1,k}, (\{A_{1,k}/A_{2,k}, B_{1,k}/B_{2,k}\} \rightarrow D_{1,k}/D_{3,k}))$ |
| (5*a*) $(C/R_{1,k}, (\{D_{1,k}/D_{2,k}\} \rightarrow E_{1,k}/E_{2,k}))$ | (5*b*) $(C/R_{1,k}, (\{D_{1,k}/D_{3,k}\} \rightarrow E_{1,k}/E_{2,k}))$ |

The algorithm was implemented in Java and the tests have been done on an Intel Quad Core i3-2310M with 2.10GHz and 8GB of memory. We have used three scenarios for performing our tests.

In the first scenario we examine the influence of the size of $\mathcal{F}_1$ and $\mathcal{F}_2$ on the execution time of Algorithm 1. We have used $\mathcal{F}_1^j$ and $\mathcal{F}_2^j$, such that $1 \leq j \leq 45$. Figure 4 shows reasonable execution time (approximately 2 minutes) for computing $cover\mathcal{F}$ from sets of XFD $\mathcal{F}_1$ and $\mathcal{F}_2$ where $|\mathcal{F}_1| + |\mathcal{F}_2| = 450$. Figure 4 also shows how $cover\mathcal{F}$ increases: at each step as we add 25 XFD to $|\mathcal{F}_1| + |\mathcal{F}_2|$, set $cover\mathcal{F}$ has about 50 XFD more than its previous version.

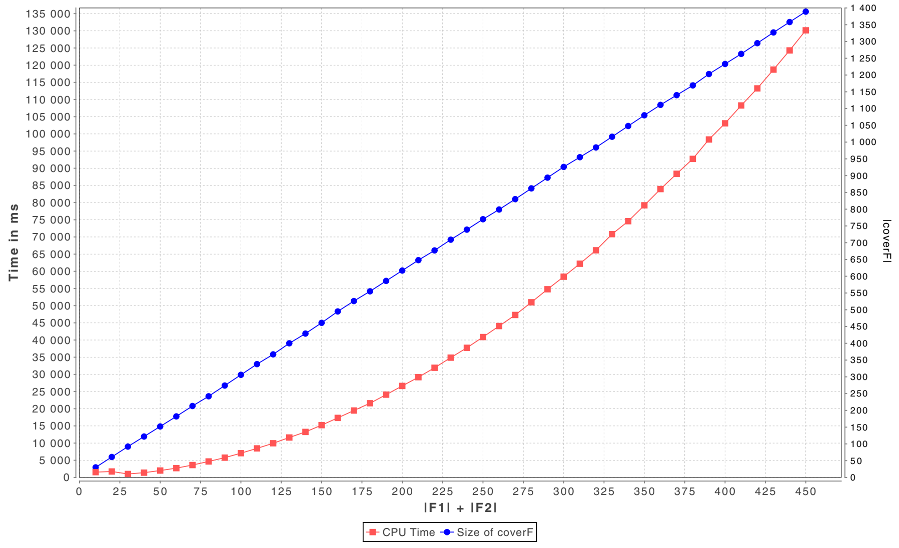

**Fig. 4.** Scenario 1: CPU time for the computing of $cover\mathcal{F}$ and the evolution of its size

In the second scenario we examine again the influence of the size of $\mathcal{F}_1$ and $\mathcal{F}_2$ on the execution time of Algorithm 1. Notice that, in the first scenario, the functional dependencies involving index $k$ concerns only one subtree. In this second scenario, we allow an XFD involving index $k = 1$ to derive an XFD involving index $k = 2$, and so on. To do this, we add to $\mathcal{F}_1^j$ (resp. $\mathcal{F}_2^j$) the XFD of the form $(7a)$ $(C, (\{R_{1,k}/E_{1,k-1}/E_{2,k-1}\} \rightarrow R_{1,k}/D_{1,k}/D_{2,k}))$, resp. $(7b)$ $(C, (\{R_{1,k}/E_{1,k-1}/E_{2,k-1}\} \rightarrow R_{1,k}/D_{1,k}/D_{3,k}))$, with $2 \leq k \leq j$.

As shown in Figure 5, the execution time for computing $cover\mathcal{F}$ is more important than the one obtained with the first scenario. For instance, for sets $\mathcal{F}_1$ and $\mathcal{F}_2$ (such that $|\mathcal{F}_1|+|\mathcal{F}_2| = 262$) we need 53 minutes to compute $cover\mathcal{F}$. This behaviour is explained by two facts:

- XFD of the form (7*a*) and (7*b*) are not added to $cover\mathcal{F}$ due to the condition in line 9 of Algorithm 1. Checking this condition is an expensive task because the computation of $(C, R_{1,k}/E_{1,k-1}/E_{2,k-1})^+_{\mathcal{F}_i}$ involves many paths.
- For this example, lines 12-15 of Algorithm 1 generate many XFD dramatically increasing the number of XFD in $cover\mathcal{F}$. Indeed, $|cover\mathcal{F}|$ has about 10610 XFD when $|\mathcal{F}^j_1| + |\mathcal{F}^j_2|$ is 262.

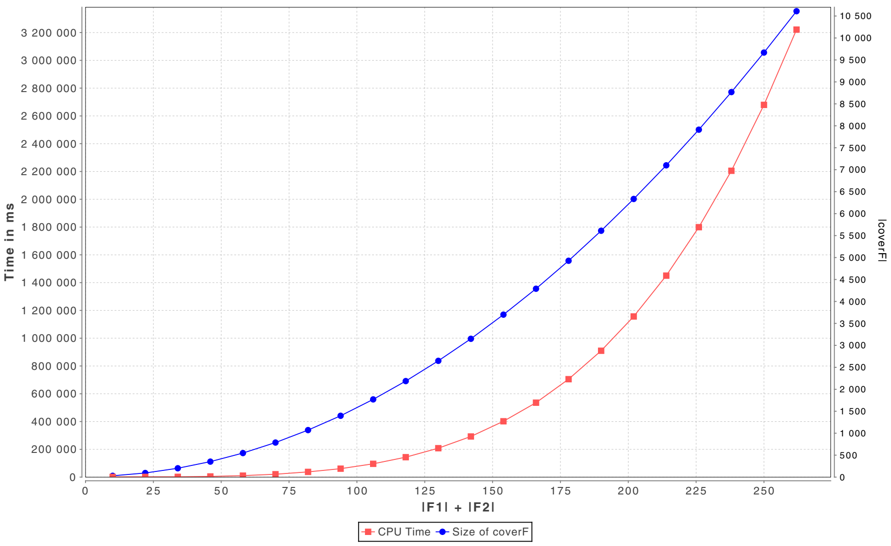

**Fig. 5.** Scenario 2: CPU time for the computing of $cover\mathcal{F}$ and the evolution of its size

In the third scenario, we compare the algorithm built to compute $\mathcal{F} = (\mathcal{F}^+_1 \cap \mathcal{F}^+_2) \cup \mathcal{K}_1 \cup \mathcal{K}_2$ from $\mathcal{F}^+_1$ and $\mathcal{F}^+_2$ (presented in [2]) with Algorithm 1 (which computes $cover\mathcal{F}$). Recall that in Section 7, we have shown that $cover\mathcal{F}$ is equivalent to $\mathcal{F}$. Now, Table 6 compares these two algorithms. Line 1 in Table 6 shows the results with sets $\mathcal{F}^1_1$ and $\mathcal{F}^1_2$ while line 4 shows the result with $\mathcal{F}^2_1$ and $\mathcal{F}^2_2$, and line 5 shows the result with $\mathcal{F}^3_1$ and $\mathcal{F}^3_2$, the same sets used in scenario 1. When computing the set $\mathcal{F}$ for sets $\mathcal{F}^3_1$ and $\mathcal{F}^3_2$ with the algorithm in [2], we obtain an *out-of-memory* error after 5 minutes. For the same sets of XFD, Algorithm 1 takes approximately 2.9 seconds and $|cover\mathcal{F}| = 92$. Since the test concerning line 5 does not produce a result for the algorithm in [2], we perform tests of line 2 and 3 on a modified tree, *i.e.*, on $\mathcal{T}$ without the leaves. In other words, we delete nodes $A_{2,2}$, $B_{2,2}$, $D_{2,2}$, $D_{3,2}$ and $E_{2,2}$ from a tree $\mathcal{T}$ with $k = 2$ sub-trees. The tree considered in line 3 contains nodes $F_{1,2}$ and $G_{1,2}$ in addition to nodes in the tree considered in line 2. As expected, in all cases, Algorithm 1 is much more efficient than the algorithm in [2]. Moreover, the size of $\mathcal{F}$ grows dramatically while the size of $cover\mathcal{F}$ increases slightly.

**Table 6.** Comparison: $t_1$ is the time needed to compute $\mathcal{F}$ and $t_2$ is the time needed to compute $cover\mathcal{F}$

| | $\|\mathbb{P}_1\| + \|\mathbb{P}_2\|$ | $\|\mathcal{F}_1\|$ | $\|\mathcal{F}_2\|$ | $\|\mathcal{F}_1^+\|$ | $\|\mathcal{F}_2^+\|$ | $\|\mathcal{F}\|$ | $t_1$ (ms) | $\|cover\mathcal{F}\|$ | $t_2$ (ms) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 12 | 5 | 5 | 3 835 | 3 835 | 5 755 | 19 211 | 30 | 160 |
| 2 | 17 | 7 | 7 | 3 865 | 3 865 | 5 785 | 24 401 | 32 | 195 |
| 3 | 18 | 8 | 8 | 3 928 | 3 928 | 5 911 | 26 476 | 34 | 204 |
| 4 | 23 | 10 | 10 | 7 670 | 7 670 | 11 510 | 165 460 | 61 | 503 |
| 5 | 34 | 15 | 15 | ? | ? | ? | $>$ 5min | 92 | 2 949 |

Our experiments confirm the time complexity presented in Section 7.3, and reinforce the importance of computing the smaller set $cover\mathcal{F}$ instead of the equivalent set $\mathcal{F}$ considered in [2]. The worst case happens when documents have many equivalent paths and derivations that involve a lot of paths. We have used the closure algorithm to test, successfully, the equivalence between $\mathcal{F}$ and $cover\mathcal{F}$ on several examples. These tests contribute to the validation of the correctness of our method.

## 9 Conclusions

We are motivated by applications on a multiple system environment and we have presented a method for establishing the biggest set of XFD that can be satisfied by any document conceived to respect local XFD. One important originality of our work is the fact that we do not deal with data, only with the available constraints (XFD in our case). Our approach is not only interesting for multiple system applications, but also in a conservative constraint evolution perspective. To reach our goals, a new axiom system, built for XFD defined over a context and two kinds of equality, was introduced and proved to be sound and complete.

As some future directions that follow from this work, we mention:

– By using the schema evolution method of [9] together with our computation of $cover\mathcal{F}$, the generation of a new type and a new set of integrity constraints that will allow interoperability without abolishing constraint verification. We are currently working on a platform that puts together these tools.
– The extension of our method to other kinds of integrity constraints such as inclusion constraints.
– An incremental computation of $cover\mathcal{F}$, following the evolution of local constraints or systems.
– The detection of local XFD that are not selected in $cover\mathcal{F}$ but that could be included in it by correcting the associated documents that do not respect them.
– The implementation of an XFD validator over the local systems, in a map-reduce approach, by considering the set $cover\mathcal{F}$ as the set of constraints that should be respected by the data of our multiple system.

## A Soundness of the Axiom System

In this section we prove that our axiom system is sound, *i.e.*, our axioms always lead to true conclusions when we deal with complete XML trees. We start by proving some lemmas. The first one deals with properties concerning the longest common prefix of paths. The following example illustrates the situation it concerns.

*Example 7.* We consider the XML document in Figure 2 and the following paths: $P_K = univ/undergraduate/@domain$, $P_J = univ/undergraduate/students/student/idSt$ and $P_I = univ/undergraduate/students/student/nameSt$. In this situation we have $P_I \cap P_J = univ/undergraduate/students/student$ and $P_J \cap P_K = univ/undergraduate$. Clearly, $P_J \cap P_K \preceq P_I \cap P_J$. Then, consider path instances where $isInst\_lcp(P_I, I, P_J, J) = true$ and $isInst\_lcp(P_I, I, P_K, K) = true$. For instance, let instance $K = \epsilon/2/2.1$, instance $J = \epsilon/2/2.2/2.2.0/2.2.0.0$ and $I = \epsilon/2/2.2/2.2.0/2.2.0.1$. Notice that in this case we also have: $isInst\_lcp(P_J, J, P_K, K) = true$. □

The above example suggests that a kind of transitivity property could be established for the function *isInst_lcp*. The following lemma proves that this is actually possible.

**Lemma 3.** Let $\mathcal{T}$ be an XML document and $\mathbb{P}$ its associated set of simple paths. Let $P_I$, $P_J$, $P_K$ be distinct paths in $\mathbb{P}$. If $P_J \cap P_K \preceq P_I \cap P_J$ and $isInst\_lcp(P_I, I, P_J, J) = isInst\_lcp(P_I, I, P_K, K) = true$ then $isInst\_lcp(P_J, J, P_K, K) = true$. □

The next example illustrates a special situation where an XFD not satisfied by a given document has at the left-hand side a path which is a prefix of the path on the right-hand side.

*Example 8.* We consider the example in Figure 2, the XFD $f = (univ/undergradute, (\{courses//titleC, courses//prerequisitesC\} \rightarrow courses//prerequisitesC/codeC))$ and the branching path $M$ defined by $f$. The document of Figure 2 does not satify $f$. Notice that $P_2 = univ/undergradute/courses/course/prerequisitesC$ in the left-hand side of $f$ is a prefix for $P = univ/undergradute/courses/course/prerequisitesC/codeC$ in right-hand side of $f$. Also remark that we can find two projections of the XML tree over $M$ such that $Last(\Pi^1_M(\mathcal{T})[C/P_2]) =_N Last(\Pi^2_M(\mathcal{T})[C/P_2])$: the two projections ending on node 2.3.0.2. □

The following lemma proves that in situations as the one illustrated by Example 8 we can always find two projections of the XML tree over the branching path $M$ such that $Last(\Pi^1_M(\mathcal{T})[C/P_j]) =_N Last(\Pi^2_M(\mathcal{T})[C/P_j])$, where $P_j$ is the path on the left-hand side which is a prefix of the one on the right-hand side.

**Lemma 4.** Let $\mathcal{T}$ be an XML document, $f = (C, (\{P_1\,[E_1], \ldots, P_n\,[E_n]\} \rightarrow P_{n+1}\,[E_{n+1}]))$ an $XFD$ and let $M$ be the branching path $\{C/P_1, \ldots, C/P_{n+1}\}$. If $\mathcal{T} \not\models f$ and there exists a $j \in [1 \ldots n]$ such that $P_j \preceq P_{n+1}$ then we can find two projections $\Pi^1_M(\mathcal{T})$ and $\Pi^2_M(\mathcal{T})$ for $M$ in $\mathcal{T}$ such that $Last(\Pi^1_M(\mathcal{T})[C/P_j]) =_N Last(\Pi^2_M(\mathcal{T})[C/P_j])$. □

In this appendix we just show the soundness proof of axiom A4.

**Theorem 2.** *Axiom A4 is sound for XFD on complete XML trees.* □

*Proof*: We consider a complete XML document $\mathcal{T} = (t, type, value)$.
A4: Let $f = (C, (\{P'_1\,[E'_1], \ldots, P'_n\,[E'_n]\} \rightarrow P_{n+1}\,[E_{n+1}]))$ and $f' = (C, (\{P_1\,[E_1], \ldots, P_n\,[E_n]\} \rightarrow P_{n+1}\,[E_{n+1}]))$. The proof is by contradiction. Suppose that $\mathcal{T} \models f$ but $\mathcal{T} \not\models f'$. From Axiom $A1$, we can assume that for all $i \in [1 \ldots n]$, $P_i \neq P_{n+1}$. From Definition 5, we can deduce that there exist two projections $\Pi^1_M(\mathcal{T})$ and $\Pi^2_M(\mathcal{T})$ for the branching path $M = \{C/P_1, \ldots, C/P_{n+1}\}$ in $\mathcal{T}$ such that $\tau^1[C/P_{n+1}] \neq_{E_{n+1}} \tau^2[C/P_{n+1}]$ and $\tau^1[C/P_1, \ldots, C/P_n] =_{E_i, i \in [1 \ldots n]} \tau^2[C/P_1, \ldots, C/P_n]$. We now show that there exist two projections $\Pi^1_{M'}(\mathcal{T})$ and $\Pi^2_{M'}(\mathcal{T})$, constructed from $\Pi^1_M(\mathcal{T})$ and $\Pi^2_M(\mathcal{T})$, for the branching path $M' = \{C/P'_1, \ldots, C/P'_n, C/P_{n+1}\}$ in $\mathcal{T}$ such that:

$$u^1[C/P'_1, \ldots, C/P'_n] =_{E'_i, i \in [1 \ldots n]} u^2[C/P'_1, \ldots, C/P'_n] \text{ and} \tag{1}$$

$$u^1[C/P_{n+1}] \neq_{E_{n+1}} u^2[C/P_{n+1}]. \tag{2}$$

However, from our hypothesis we know that for all two projections $\Pi^1_{M'}(\mathcal{T})$ and $\Pi^2_{M'}(\mathcal{T})$ such that (1) is satisfied then we have $u^1[C/P_{n+1}] =_{E_{n+1}} u^2[C/P_{n+1}]$. If $\Pi^1_{M'}(\mathcal{T})$ and $\Pi^2_{M'}(\mathcal{T})$ really exist, we have a contradiction with (2) and the axiom $A4$ will be satisfied.

The proof is by showing that it is possible to obtain two projections for $M'$ satisfying (1) and (2). We start by considering that $\Pi^1_{M'}(\mathcal{T})[C/P_i] = \Pi^1_M(\mathcal{T})[C/P_i]$ and $\Pi^2_{M'}(\mathcal{T})[C/P_i]) = \Pi^2_M(\mathcal{T})[C/P_i]\ \forall\, i \in [1 \ldots n+1]$.

1. If $\exists\, k \in [1 \ldots n]$ such that $P_k \preceq P_{n+1}$ (Figure 6(a)) then, from Lemma 4, we can consider that :

$$Last(\Pi^1_{M'}(\mathcal{T})[C/P_k]) =_N Last(\Pi^2_{M'}(\mathcal{T})[C/P_k]). \tag{3}$$

   Since $t$ is complete there exist instances $J_i$ such that $\forall\, i \in [1 \ldots n]$, $\Pi^1_{M'}(\mathcal{T})[C/P_i] \preceq J_i$ and $J_i \in Instances(C/P'_i, t)$ (see Figure 6(a)). Let $\forall\, i \in [1 \ldots n]$, $\Pi^1_{M'}(\mathcal{T})[C/P'_i] = \Pi^2_{M'}(\mathcal{T})[C/P'_i] = J_i$. Then by considering these instances $J_i$ for paths $C/P'_i$ and by using Lemma 3, we can show that $\forall\, i, j \in [1 \ldots n+1]$ (recall that we consider that $P'_{n+1}{=}P_{n+1}$):

$$isInst\_lcp(C/P'_i, \Pi^1_{M'}(\mathcal{T})[C/P'_i], C/P'_j, \Pi^1_{M'}(\mathcal{T})[C/P'_j]) = true \tag{4}$$

$$\text{and } isInst\_lcp(C/P'_i, \Pi^2_{M'}(\mathcal{T})[C/P'_i], C/P'_j, \Pi^2_{M'}(\mathcal{T})[C/P'_j]) = true. \tag{5}$$

   Thus, in this case, it is possible to have projections $\Pi^1_{M'}(\mathcal{T})$ and $\Pi^2_{M'}(\mathcal{T})$ satisfying 1 and 2.

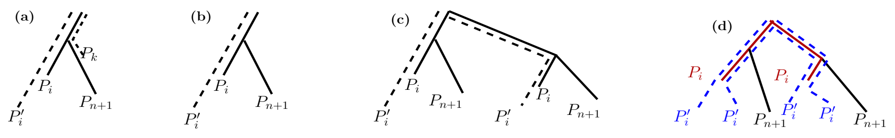

**Fig. 6.** Graphical representation of paths and possible projections. Case (a) Path $P_k \prec P_{n+1}$ where $P_k$ is one of the paths in the left-handside of XFD $f'$ and $P_i \preceq P'_i$. Case (b): Node equality for last nodes in $P_i$. Both projections $\Pi^1_{M'}(\mathcal{T})$ and $\Pi^2_{M'}(\mathcal{T})$ have the same instance for path $P'_i$. Case (c) and (d): Value equality for last nodes in $P_i$. In case (c) there is only one instance of $P'_i$ ($P_i \preceq P'_i$). In case (d) there are two instances of $P'_i$ ($P_i \preceq P'_i$). □

2. Otherwise if $\forall\, i \in [1 \dots n]$, $P_i \npreceq P_{n+1}$ and $P_i \preceq P'_i$ we can have the following situations:
   (a) If we consider **node equality**, we have $Last(\Pi^1_{M'}(\mathcal{T})[C/P_i]) =_N Last(\Pi^2_{M'}(\mathcal{T})[C/P_i])$ (Figure 6(b)). Since $t$ is complete there exists an instance $J_i$ such that $\Pi^1_{M'}(\mathcal{T})[C/P_i] \preceq J_i$ and $J_i \in Instances(C/P'_i, t)$. Let $\Pi^1_{M'}(\mathcal{T})[C/P'_i] = \Pi^2_{M'}(\mathcal{T})[C/P'_i] = J_i$.
   (b) If we consider **value equality**, we have $Last(\Pi^1_{M'}(\mathcal{T})[C/P_i]) =_V Last(\Pi^2_{M'}(\mathcal{T})[C/P_i])$. Since $t$ is complete there exist instances $J^1_i$, $J^2_i$ such that $\Pi^1_{M'}(\mathcal{T})[C/P_i] \preceq J^1_i$, $\Pi^2_{M'}(\mathcal{T})[C/P_i] \preceq J^2_i$ and $J^1_i, J^2_i \in Instances(C/P'_i, t)$.
      – If $Last(J^1_i) =_V Last(J^2_i)$ then let $\Pi^1_{M'}(\mathcal{T})[C/P'_i] = J^1_i$ and $\Pi^2_{M'}(\mathcal{T})[C/P'_i] = J^2_i$ (Figure 6(c)).
      – Otherwise if $Last(J^1_i) \neq_V Last(J^2_i)$ then, since $P_i \preceq P'_i$ and $Last(\Pi^1_{M'}(\mathcal{T})[C/P_i]) =_V Last(\Pi^2_{M'}(\mathcal{T})[C/P_i])$, there exists two instances $J^3_i$, $J^4_i$ such that $\Pi^1_{M'}(\mathcal{T})[C/P_i] \preceq J^3_i$, $\Pi^2_{M'}(\mathcal{T})[C/P_i] \preceq J^4_i$ and $J^3_i, J^4_i \in Instances(C/P'_i, t)$, $Last(J^1_i) =_V Last(J^4_i)$ and $Last(J^2_i) =_V Last(J^3_i)$. In this case, let $\Pi^1_{M'}(\mathcal{T})[C/P'_i] = J^1_i$ and $\Pi^2_{M'}(\mathcal{T})[C/P'_i] = J^4_i$ (Figure 6(d)).

   Then by considering these instances $J_i$ for paths $C/P'_i$ and by using Lemma 3, we can show (4) and (5) in each case. Since $\Pi^1_{M'}(\mathcal{T})$ and $\Pi^2_{M'}(\mathcal{T})$ exist and conditions (1), (2) are satisfied, we can conclude that $A4$ is sound.

## B Completeness of the Axiom System

Before tackling the completeness issue, it is important to show the central fact about the closure of a set of paths. It enables us to tell on a glance whether an XFD follows from a set $\mathcal{F}$ by the axiom system. The next lemma tells us how.

**Lemma 5.** Let $X = \{P_1, \dots, P_n\}$ and $Y = \{P_{n+1}, \dots, P_{n+m}\}$ be two sets of paths. Let $E = (E_1, \dots, E_n)$ and $E' = (E_{n+1}, \dots, E_{n+m})$. We have $\mathcal{F} \vdash (C, (\{P_1\,[E_1], \dots, P_n\,[E_n]\} \rightarrow \{P_{n+1}\,[E_{n+1}], \dots, P_{n+m}\,[E_{n+m}]\}))$ iff $C/Y[E'] \subseteq (C, X[E])^+$. □

To prove the completeness of our axiom system, we would like to define a special tree having two instances (except for the root node) for every path $P \in \mathbb{P}$. However, the following examples show that depending on the conditions imposed on paths, it is not possible to have *two* instances for *every* path $P \in \mathbb{P}$.

*Example 9.* We want to build a complete tree having exactly two instances for each path in $\mathbb{P}$. Let us consider value equality and two paths $P$ and $Q$ such that $P \prec Q$. We denote by $I_{P_1}$ and $I_{P_2}$ the two instances of $P$ on a tree $t$. We denote by $I_{Q_1}$ and $I_{Q_2}$ the two instances of $Q$ on a tree $t$. Suppose that $Last(I_{P_1}) =_V Last(I_{P_2})$ and $Last(I_{Q_1}) \neq_V Last(I_{Q_2})$. Based on this situation, the functional dependency $P \to Q$ is not satisfied by this tree. Then, we can apply Lemma 4, to conclude that there is an instance of $P$ which is a prefix of both (distinct) instances of $Q$. As we want just two instances for each path, to have two instances of $Q$ we should have $Last(I_{P_1}) =_N Last(I_{P_2})$. In other words, in this situation, we cannot have a tree with two instances for $P$. Indeed, Figure 7 illustrates that a tree having two instances of $P$ and respecting the constraints $Last(I_{P_1}) =_V Last(I_{P_2})$ and $Last(I_{Q_1}) \neq_V Last(I_{Q_2})$ must have four instances of $Q$.

Now let us consider that a node equality condition is imposed on the instances of a path $P$. In this situation we have $Last(I_{P_1}) =_N Last(I_{P_2})$. Clearly, in this case, $P$ has only one instance. □

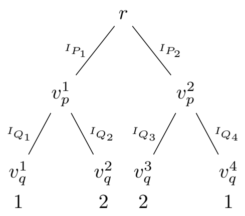

**Fig. 7.** An XML tree with two instances value equal for the path $P$ and four instances for the path $Q$ with $I_{P_1}$ prefix of $I_{Q_1}$, $I_{Q_2}$ and $I_{P_2}$ prefix of $I_{Q_3}$, $I_{Q_4}$

Based on Example 9, we introduce the definition of our special tree, having *at most* two instances for each path in $\mathbb{P}$.

**Definition 11 (Two-instance Tree)**
*Let $\mathcal{F}$ be a set of XFD. Let $\mathcal{T} = (t, type, value)$ be an XML document where the tree $t$, built according to the construction properties below, is called two-instance tree. Let $\mathbb{P}$ be the set of paths associated to $\mathcal{T}$, let $X \subseteq \mathbb{P}$, and let $E = (E_1, \ldots, E_n)$ be the equality list associated to $X$. We denote by $|Instances(P, t)|$ the number of instances of a path $P$ in $t$.*

CONSTRUCTION PROPERTIES:

1. *For each $P \in I\!P$, $|Instances(P,t)|$ is at most 2 ($I_1$ or $I_2$) and when $|Instances(P,t)| = 2$:*
   *(a) we have $C/P[V] \in (C, X[E])^+$ iff $Last(I_1) =_V Last(I_2)$;*
   *(b) we have $C/P[E'] \notin (C, X[E])^+$ iff $Last(I_1) \neq_{E'} Last(I_2)$*
2. *For each $P \in I\!P$, $|Instances(P,t)| = 2$ except when:*
   *(a) $Last(P)$ is the root of $t$, or*
   *(b) by considering value equality, $|Instances(P,t)| = 2$ provokes the violation of condition 1 for another path $Q \in I\!P$ with $P \prec Q$, or*
   *(c) $X[E] \rightarrow P[N]$ or*
   *(d) $Last(P) \in \Sigma_{att}$ and $Parent(P)$ verifies condition 2a, or 2b or 2c.* □

**Lemma 6.** Let $\mathcal{F}$ be a set of XFD. Let $\mathcal{T} = (t, type, value)$ be an XML document where $t$ is a two-instance tree. Let $\mathbb{P}$ be the set of paths associated to $\mathcal{T}$ and let $X \subseteq \mathbb{P}$. The following properties hold for $t$:

1. If $C/P \in \mathbb{P}$ and $|Instances(C/P,t)| = 1$ then $C/P[E'] \in (C, X[E])^+$.
2. If $P, Q \in \mathbb{P}$, $P \preceq Q$ and there is an instance $I_P \in Instances(P,t)$, and instances $I_{Q_1}$ and $I_{Q_2} \in Instances(Q,t)$ such that $I_P \preceq I_{Q_1}$ and $I_P \preceq I_{Q_2}$ then $|Instances(P,t)| = 1$.
3. If $C/P \in \mathbb{P}$ then $C/P[E'] \in (C, X[E])^+$ iff $\mathcal{T} \models (C, (X[E] \rightarrow P[E']))$. □

We now prove that the axiom system introduced in Definition 8 is complete. In other words, given a set of XFD $\mathcal{F}$, by using our inference rules, we can derive all XFD $f$ such that $\mathcal{F} \models f$.

**Theorem 3.** *If $\mathcal{F} \models f$ then $\mathcal{F} \vdash f$.* □

*Proof:* The proof is by contrapositive: we show that if $\mathcal{F} \not\vdash f$ then $\mathcal{F} \not\models f$.
Let $f = (C, (\{P_1\,[E_1], \ldots, P_n\,[E_n]\} \rightarrow \{P_{n+1}\,[E_{n+1}] \ldots P_{n+m}[E_{n+m}]\}))$. Then, we consider that $X = \{C/P_1, \ldots, C/P_n\}$, $Y = \{C/P_{n+1}, \ldots, C/P_{n+m}\}$ and that both $X$ and $Y$ are in a given $\mathbb{P}$. Let $E = (E_1, \ldots, E_n)$.
If $\mathcal{F} \not\models f$ then there must be an XML document that satisfies $\mathcal{F}$ but does not satisfy $f$. The proof consists in showing the existence of such a document.

Let us suppose an XML document $\mathcal{T} = (t, type, value)$ where $t$ is a two-instance tree defined on the set of paths $X = \{C/P_1, \ldots, C/P_n\}$.

**Fact 1:** $\mathcal{T} \models \mathcal{F}$
The proof is by contradiction. We suppose that $\mathcal{T} \not\models g$, where $g$ is an XFD $(C,(\{Q_1\,[E'_1], \ldots, Q_k\,[E'_k]\} \rightarrow Q_{k+1}\,[E'_{k+1}]))$ in $\mathcal{F}$. From Definition 5, as $\mathcal{T} \not\models g$, we can deduce that there exist two projections $\Pi^1_M(\mathcal{T})$ and $\Pi^2_M(\mathcal{T})$ for the branching path $M = \{C/Q_1, \ldots, C/Q_{k+1}\}$ in $\mathcal{T}$ such that:

$$\tau^1[C/Q_1, \ldots, C/Q_k] =_{E'_i, i \in [1 \ldots k]} \tau^2[C/Q_1, \ldots, C/Q_k] \text{ and} \tag{6}$$

$$\tau^1[C/Q_{k+1}] \neq_{E'_{k+1}} \tau^2[C/Q_{k+1}]. \tag{7}$$

From (7) we have that $\Pi^1_M(\mathcal{T})[C/Q_{k+1}] \neq \Pi^2_M(\mathcal{T})[C/Q_{k+1}]$ and $|Instances(Q_{k+1}, t)| = 2$. From Definition 11(1), we obtain:

$$C/Q_{k+1}[E'_{k+1}] \notin (C, X[E])^+. \tag{8}$$

From Definition 2, we know that the instances of two paths belonging to the same branching path match on their longest common prefix path. Formally, for all combination of paths $Q_i$ and $Q_j$ such that $1 \leq i \leq k+1$ and $1 \leq j \leq k+1$, we have:

<u>Considering $\Pi^1_M(\mathcal{T})$:</u>

$$isInst\_lcp(C/Q_i, \Pi^1_M(\mathcal{T})[C/Q_i], C/Q_j, \Pi^1_M(\mathcal{T})[C/Q_j]) = true \tag{9}$$

<u>Considering $\Pi^2_M(\mathcal{T})$:</u>

$$isInst\_lcp(C/Q_i, \Pi^2_M(\mathcal{T})[C/Q_i], C/Q_j, \Pi^2_M(\mathcal{T})[C/Q_j]) = true \tag{10}$$

and we can also determine the following special nodes for $1 \leq i \leq k$:

$$\begin{aligned} v^1_{i,k+1} &= Last(\Pi^1_M(\mathcal{T})[C/Q_i] \cap \Pi^1_M(\mathcal{T})[C/Q_{k+1}]) \text{ and} \\ v^2_{i,k+1} &= Last(\Pi^2_M(\mathcal{T})[C/Q_i] \cap \Pi^2_M(\mathcal{T})[C/Q_{k+1}]) \end{aligned} \tag{11}$$

From (9) and (10), together with the definition of *isInst_lcp* we know that positions $v^1_{i,k+1}$ and $v^2_{i,k+1}$ exist in $t$. We have to consider two cases:

(a) $v^1_{i,k+1} = v^2_{i,k+1}$ (illustrated in Figure 8(a))

(b) $v^1_{i,k+1} \neq v^2_{i,k+1}$ (illustrated in Figure 8(b))

We can easily show that it is always possible to choose for each $i \in [1 \ldots k]$, a path $C/R_i \in \mathbb{P}$ respecting the following property:

$$C/R_i[E'_i] \in (C, X[E])^+ \text{ and } Q_i \cap Q_{k+1} \preceq R_i \preceq Q_i \tag{12}$$

Now from property (12), the XFD $g = (C, (\{Q_1\,[E'_1], \ldots, Q_k\,[E'_k]\} \rightarrow Q_{k+1}\,[E'_{k+1}]))$ in $\mathcal{F}$, and the axiom Branch Prefixing (Definition 8, axiom $A4$) we deduce the XFD $g' = (C, (\{R_1\,[E'_1], \ldots, R_k\,[E'_k]\} \rightarrow Q_{k+1}\,[E'_{k+1}]))$. Next, we assume that if $C/\{R_1, \ldots, R_k\}[E'] \subseteq (C, X[E])^+$ then $C/Q_{k+1}[E'_{k+1}] \in (C, X[E])^+$. Indeed, by Definition 10 we know that $X[E] \rightarrow \{R_1\,[E'_1], \ldots, R_k\,[E'_k]\}$. From this rule and $g'$, we derive $X[E] \rightarrow Q_{k+1}[E'_{k+1}]$ by using the axiom Transitivity ($A3$). Thus, from Definition 10, we obtain $C/Q_{k+1}[E'_{k+1}] \in (C, X[E])^+$ which contradicts (8): $Q_{k+1}[E'_{k+1}] \notin (C, X[E])^+$. Thus, we conclude that $\mathcal{T} \models g$ for any $g \in \mathcal{F}$. In other words, $\mathcal{T} \models \mathcal{F}$.

**Fact 2:** $\mathcal{T} \not\models f$

Recall, from the beginning of our proof, that $f$ is the XFD $(C, (X[E] \rightarrow Y[E'']))$. As $X[E] \subseteq (C, X[E])^+, \forall\, P_i \in X$ we have $Last(\Pi^1_M(\mathcal{T})[P_i]) =_{E_i} Last(\Pi^2_M(\mathcal{T})[P_i])$ for two given projections of $M$ on $\mathcal{T}$. From our hypothesis, $\mathcal{F} \not\vdash f$ and so $C/Y[E''] \not\subseteq$

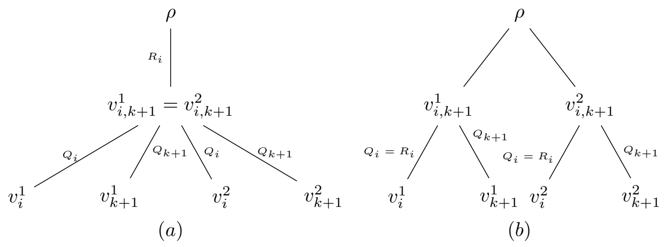

**Fig. 8.** Illustration of the two cases (a)$v^1_{i,k+1} = v^2_{i,k+1}$ and (b)$v^1_{i,k+1} \neq v^2_{i,k+1}$

$(C, X[E])^+$. Thus, there is at least one path $P \in Y$ having instances $I_1$ and $I_2$ such that $Last(I_1) \neq_E Last(I_2)$. We deduce that $\mathcal{T} \not\models f$.

In conclusion we have built a tree $\mathcal{T}$ such that $\mathcal{T} \models \mathcal{F}$ and $\mathcal{T} \not\models f$ which establishes the proof of Theorem 3. □

**Acknowledgments.** We would like to thank Françoise Gire, Hicham Idabal, Béatrice Bouchou and Martin Musicante for initial discussions that gave rise to this work.

## References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley Publishing Company (1995)
2. Amavi, J., Halfeld Ferrari, M.: An axiom system for XML and an algorithm for filtering XFD. Tech. Rep. RR-2012-03, LIFO/Université d'Orléans (2012). http://www.univ-orleans.fr/lifo/rapports.php?lang=fr&annee=2012
3. Arenas, M., Libkin, L.: A normal form for XML documents. ACM Transactions on Database Systems (TODS) 29 No.1 (2004)
4. Bouchou, B., Cheriat, A., Halfeld Ferrari, M., Laurent, D., Lima, M.A., Musicante, M.: Efficient constraint validation for updated XML databases. Informatica **31**(3), 285–310 (2007)
5. Bouchou, B., Halfeld Ferrari, M., Lima, M.: Contraintes d'intégrité pour XML. visite guidée par une syntaxe homogène. Technique et Science Informatiques **28**(3), 331–364 (2009)
6. Bouchou, B., Halfeld-Ferrari, M., Lima, M.A.V.: A Grammarware for the Incremental Validation of Integrity Constraints on XML Documents under Multiple Updates. In: Hameurlain, A., Küng, J., Wagner, R., Liddle, S.W., Schewe, K.-D., Zhou, X. (eds.) Transactions on Large-Scale Data- and Knowledge-Centered Systems VI. LNCS, vol. 7600, pp. 167–197. Springer, Heidelberg (2012)
7. Castanier, E., Coletta, R., Valduriez, P., Frisch, C.: Public data integration with websmatch. In: BDA 2012 (2012)
8. Cecchin, F., de Aguiar Ciferri, C.D., Hara, C.S.: XML data fusion. In: DaWak. pp. 297–308 (2010)
9. Chabin, J., Halfeld Ferrari, M., Musicante, M.A., Réty, P.: Conservative Type Extensions for XML Data. In: Hameurlain, A., Küng, J., Wagner, R. (eds.) TLDKS IX. LNCS, vol. 7980, pp. 65–94. Springer, Heidelberg (2013)

10. Chirkova, R., Libkin, L., Reutter, J.L.: Tractable XML data exchange via relations. In: Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011. pp. 1629–1638 (2011)
11. Crane, G.: What do you do with a million books? D-Lib Magazine 12(3) (March 2006)
12. Gire, F., Idabal, H.: Regular tree patterns: a uniform formalism for update queries and functional dependencies in XML. In: EDBT/ICDT Workshops (2010)
13. Hartmann, S., Trinh, T.: Axiomatising Functional Dependencies for XML with Frequencies. In: Dix, J., Hegner, S.J. (eds.) FoIKS 2006. LNCS, vol. 3861, pp. 159–178. Springer, Heidelberg (2006)
14. He, Q., Ling, T.W.: Extending and inferring functional dependencies in schema transformation. In: Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management. pp. 12–21 (2004)
15. Kot, Ł., White, W.: Characterization of the Interaction of XML Functional Dependencies with DTDs. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 119–133. Springer, Heidelberg (2006)
16. Li Lee, M., Ling, T.-W., Low, W.L.: Designing Functional Dependencies for XML. In: Jensen, C.S., Jeffery, K., Pokorný, J., Šaltenis, S., Bertino, E., Böhm, K., Jarke, M. (eds.) EDBT 2002. LNCS, vol. 2287, p. 124. Springer, Heidelberg (2002)
17. Liu, J., Vincent, M.W., Liu, C.: Functional dependencies, from relational to XML. In: Ershov Memorial Conference. pp. 531–538 (2003)
18. Pankowski, T.: Reconciling Inconsistent Data in Probabilistic XML Data Integration. In: Gray, A., Jeffery, K., Shao, J. (eds.) BNCOD 2008. LNCS, vol. 5071, pp. 75–86. Springer, Heidelberg (2008)
19. Shahriar, M.S., Liu, J.: Preserving Functional Dependency in XML Data Transformation. In: Atzeni, P., Caplinskas, A., Jaakkola, H. (eds.) ADBIS 2008. LNCS, vol. 5207, pp. 262–278. Springer, Heidelberg (2008)
20. Vincent, M.W., Liu, J., Liu, C.: Strong functional dependencies and their application to normal forms in XML. ACM Trans. Database Syst. **29**(3), 445–462 (2004)
21. Vincent, M., Liu, J., Mohania, M.: The implication problem for 'closest node' functional dependencies in complete XML documents. Journal of Computer and System Sciences **78**(4), 1045–1098 (2012)
22. Wang, J., Topor, R.: Removing XML data redundancies using functional and equality-generating dependencies. In: ADC 2005: Proceedings of the 16th Australasian database conference. pp. 65–74. Australian Computer Society Inc., Darlinghurst, Australia (2005)
23. Wu, X., Ling, T.W., Lee, M.L., Dobbie, G.: Designing semistructured databases using ORA-SS model. In: Proceedings of the 2nd International Conference on Web Information Systems Engineering, WISE (1) (2001)
24. Zhao, X., Xin, J., Zhang, E.: XML functional dependency and schema normalization. In: HIS 2009: Proceedings of the 9th International Conference on Hybrid Intelligent Systems. pp. 307–312 (2009)