Checking Satisfactions of XML Referential Integrity Constraints*

Md. Sumon Shahriar and Jixue Liu

Data and Web Engineering Lab
School of Computer and Information Science
University of South Australia, SA-5095, Australia
shamy022@students.unisa.edu.au, jixue.liu@unisa.edu.au

Abstract. Recently we proposed referential integrity constraints for XML. In defining two important referential constraints namely XML inclusion dependency and XML foreign key, we considered ordered XML data model to capture the correct semantics of data when tuples are to be produced. In this paper, we report on the performances of checking both XML inclusion dependency and XML foreign key. We show that both these constraints can be checked in linear time in the context of number of tuples and the number of paths.

1 Introduction

In relational data model, integrity constraints are well studied and established[1,2]. The important integrity constraints include key constraints, functional dependency constraints and referential integrity constraints. The constraints are used in data integrity and semantics, normalization, data integrations[3], data warehousing and data mining techniques[4].

In recent years, XML[22] is massively used as data representation and storage formation over the world wide web and together with relational data storage. Now the use of XML for many data-centric activities such as data integration, data warehousing[6,5] is note worthy. With the increasing use of XML, the research for XML in database perspective is getting much attention. One such area is the constraints mechanism for XML[7,8,9,10].

The most important constraints those are investigated in XML are XML keys[17,9,10,20], XML functional dependencies[18,19] and XML multi valued dependencies[21]. Like inclusion dependencies in relational data model, inclusion dependencies for XML is also investigated in[11,12]. In both [11,12], XML inclusion dependencies are defined in the context of XML document without XML Document Type Definition(DTD). In [11], the definition of XML inclusion dependency can't capture the correct semantics if the number of paths is more than two and later this problem is addressed in the report[12].

* This research supported with Australian Research Council(ARC) Discovery Project(DP) Fund.

J. Liu et al. (Eds.): AMT 2009, LNCS 5820, pp. 148–159, 2009.

Recently, the referential integrity constraints for XML namely XML inclusion dependency(XID) and XML foreign key(XFK)[13] are defined on the XML DTD[22]. Though the DTD has the ID and IDREF definition for denoting key and referential integrity, the drawbacks of ID and IDREF are well recognized as their scope is the entire document and behave as object identifier. When defining XFK, we use the XID and the definition of XML key proposed in [14]. The satisfactions of XID, XML key and XFK are defined on the ordered model of XML document. We use the concept **tuple** that produces semantically correct values when the satisfactions for both XID and XFK are checked. This property is not achievable from both ID and IDREF of XML DTD and Key and $KeyRef$ of XML Schema[23].

In this paper, we study the implementation and performance of checking XID and XFK proposed in [13]. The study is motivated by the fact that like in the relational database, the implementation of XFKs is critical to the quality of data in the XML database. Every time when there is a new instance for the database, we like to check the constraints against the new instance to ensure proper data is added or removed from the database. At the same time, the performance of the implementation is important to the efficiency of the database. Different ways of implementing the same mechanism will result in different performances. To a database management system, the efficiency of all processes is always critical to the success of the system.

In the literature, the checking of XML key and foreign keys using SAX was studied in [15] based on the proposal presented in [17]. An indexing based on paths on key is used in checking and the performance was shown as linear. Another study in [16] showed the XML key satisfaction checking using XPath based on DOM[24]. The study showed the checking of XML key can be done in polynomial time. We also use DOM(contrasting the use of SAX in [15]) for parsing XML document, but our implementation is different from the studies [15,16] because we use a novel method of pairing the close values of elements to capture the correct semantics in tuple generation while parsing the document.

This paper is organized as follows. In Section 2, we give the basic definitions and notation of XID and XFK. The performances of checking XID and XFK are given in Section 3 and Section 4 respectively. In Section 5, we conclude with some remarks.

2 Basic Definitions and Notation

In this section, we review some basic definitions and notation proposed in[13] that is critical to guarantee the self-containment of the paper. We first introduce the DTD and paths on DTD using examples.

A DTD is defined in our notation as $D = (EN, \beta, \rho)$ where EN contains element names, ρ is the root of the DTD and β is the function defining the types of elements. For example, the DTD D in Fig. 1(a) is represented in our notation as $\beta(A) = [U^+ \times W^+]$, $\beta(U) = [B \times M^+ \times E]$, $\beta(M) = [C^* \times D^*]$, $\beta(W) = [F \times N^+ \times I]$, $\beta(N) = [G^* \times H^*]$, $\beta(B) = \beta(C) = \beta(D) = \beta(E) = Str$, $\beta(F) = \beta(G) = \beta(H) =$

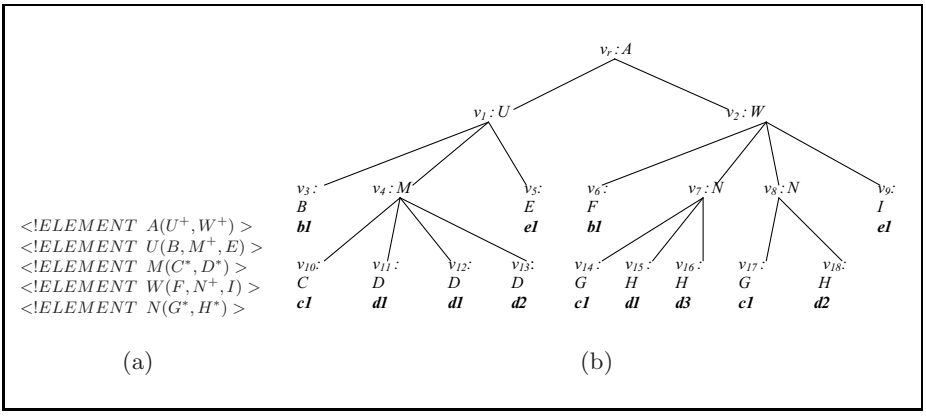

Fig. 1. (a) An XML DTD D and (b)An XML document T

$\beta(I) = Str$ where Str is $\#PCDATA$, $EN = \{A, U, B, C, D, E, M, W, F, G, H, I, N, Str\}$, and $\rho = A$. An element name and a pair of squared brackets '[ ]' each, with its multiplicity, is called a component. For example, $[U^+{}_\times W^+]$, $[B_\times M^+{}_\times E]$ are two components. A conjunctive or disjunctive sequence of components, often denoted by g, is called a structure. For example, the structure $g_1 = [B_\times M^+{}_\times E]$ is a conjunctive sequence and $g_1 = [A|B]^+$ is a disjunctive sequence. A structure is further decomposed into substructures such as g_1 is decomposed into g_a and g_b where $g_a = B^+$ and $g_b = M^+{}_\times E$.

Now we define paths on the DTD. In Fig. 1(a), we say $U/M/C$ is a simple path and $A/U/M/C$ is a complete path. A complete path starts with the root element of the DTD. The function $beg(A/U/M)$ returns A, $last(A/U/M/C)$ returns C and $par(M)$ returns U.

We now define the XML inclusion dependency.

Definition 1 (XML Inclusion Dependency). *An XML inclusion dependency over the DTD D can be defined as $\Upsilon(Q, (\{P_1, \cdots, P_n\} \subseteq \{R_1, \cdots, R_n\}))$ where Q is a complete path called* **selector***, P_i is a simple path called* **dependent** *path where $\beta(last(P_i)) = Str$ and R_i is a simple path called a* **referenced** *path where $\beta(last(R_i)) = Str$, Q/P_i and Q/R_i are complete paths.*

An XID following the above definition is valid, denoted as $\Upsilon \sqsubset D$.

For example, consider the XID $\Upsilon(A, (\{U/B, U/M/C, U/M/D, U/E\} \subseteq \{W/F, W/N/G, W/N/H, W/I\}))$ on the DTD D in Figure 1(a). The XID follows the definition1.

Before defining XID satisfaction, we introduce some definitions and notation using examples.

An XML document is represented as an XML tree $T = (v : e\ (T_1 T_2 \cdots T_f))$ if element e encloses other elements or $T = (v : e : txt)$ if e encloses the string value txt where v is the node identifier which is omitted when the context is clear, e is the element name labeled on the node, $T_1 \cdots T_f$ are subtrees. For example,

in Fig. 1(b), the document T is represented as $T_{v_r} = (v_r : A(T_{v_1}T_{v_2}))$. Then $T_{v_1} = (v_1 : U(T_{v_3}T_{v_4}T_{v_5}))$, $T_{v_2} = (v_2 : A(T_{v_6}T_{v_7}T_{v_8}T_{v_9}))$, $T_{v_3} = (v_3 : B : b1), \cdots$ Other subtrees can be expressed in the same way. We say $T_{v_{11}} =_v T_{v_{12}}$ where $T_{v_{11}} = (v_{11} : D : d1)$ and $T_{v_{12}} = (v_{12} : D : d1)$.

Now we give an example to show the important concept *hedge* which is a sequence of adjacent subtrees for a type structure. Consider the structure $g_1 = [C^* \times D^*]$ in Fig. 1(a). The trees $T_{v_{10}}T_{v_{11}}T_{v_{12}}T_{v_{13}}$ form a hedge conforming to g_1 under node v_4. However, when we consider $g_2 = [G^* \times H^*]$, there are two sequence conforming to g_2: $T_{v_{14}}T_{v_{15}}T_{v_{16}}$ for node v_7 and $T_{v_{17}}T_{v_{18}}$ for node v_8. To reference various structures and their conforming sequences, we introduce the concept *hedge*, denoted by H^g, which is a sequence of trees conforming to the structure g.

Now we introduce two concepts *minimal structure* and *minimal hedge*. A minimal structure is the one that encloses two or more given elements with the bracket '[]'. Consider $\beta(A) = [B \times M^+ \times E]$ and $\beta(M) = [C^* \times D^*]$ for D in Fig. 1(a). The minimal structure of B and C is $g_3 = [B \times M^+ \times E]$ meaning that both elements B and E is encompassed within the outermost '[]' bracket and C in $\beta(M)$. Thus the minimal hedge conforming to g_3 is $H_1^{g_3} = T_{v_3}T_{v_4}T_{v_5}$ for node v_1 in the document T in Fig. 1(b). But the minimal structure of C and D is $g_2 = [C^* \times D^*]$. So the the minimal hedges conforming to g_2 are $H_1^{g_2} = T_{v_{10}}T_{v_{11}}T_{v_{12}}T_{v_{13}}$ for node v_4 in T.

We then use minimal structure and minimal hedge to produce *tuple* for the paths in $\Upsilon(Q, (\{P_1, \cdots, P_n\} \subseteq \{R_1, \cdots, R_n\}))$. We say P(**P-tuple**) for paths P and R(**R-tuple**) for paths R. Consider an XID $\Upsilon(A, (\{U/B, U/M/C, U/M/D, U/E\} \subseteq \{W/F, W/N/G, W/N/H, W/I\}))$ on the DTD D in Fig. 1(a). Here, the selector path is A, the paths are $U/B, U/M/C, U/M/D, U/E$ are dependent paths(P-paths) and $W/F, W/N/G, W/N/H, W/I$ are referenced paths(R-paths).

First we produce the P-tuples. The minimal structure for element names B(note that $last(U/B) = B$), C, D and E is $g_4 = [B \times M^+ \times E]$ because B and E are enclosed in the '[]' and $\beta(M) = [C^* \times D^*]$. The minimal hedges for g_4 is $H_1^{g_4} = T_{v_3}T_{v_4}T_{v_5}$ under node v_1. The P-tuples for the hedge $H_1^{g_4}$ are $F_1[P] = (T_{v_3}T_{v_{10}}T_{v_{11}}T_{v_5}) = ((v_3 : B : b1)(v_{10} : C : c1)(v_{11} : D : d1)(v_5 : E : e1))$, $F_2[P] = (T_{v_3}T_{v_{10}}T_{v_{12}}T_{v_5}) = ((v_3 : B : b1)(v_{10} : C : c1)(v_{12} : D : d1)(v_5 : E : e1))$, $F_3[P] = (T_{v_3}T_{v_{10}}T_{v_{13}}T_{v_5}) = ((v_3 : B : b1)(v_{10} : C : c1)(v_{13} : D : d2)(v_5 : E : e1))$. We say the P-tuples $F_1 =_v F_2$ because the values for corresponding trees in the tuples are the same.

We now produce R-tuples. The minimal structure for element names F(note that $last(W/F) = F$), G, H and I is $g_5 = [F \times N^+ \times I]$ because the elements F and I are in the '[]' and $\beta(N) = [G^* \times H^*]$. The minimal hedge for g_5 is $H_1^{g_5} = T_{v_6}T_{v_7}T_{v_8}T_{v_9}$ for node v_2. We observe that there are two hedges for $[G^* \times H^*]$ in node v_1. Thus we first need to produce pair-wise values for G, H elements and then we need to combine those pair-wise values with F and I elements to make the semantics correct. Thus R-tuples are $F_1[R] = (T_{v_6}T_{v_{14}}T_{v_{15}}T_{v_9}) = ((v_6 : F : b1)(v_{14} : G : c1)(v_{15} : H : d1)(v_9 : I : e1))$, $F_2[R] = (T_{v_6}T_{v_{14}}T_{v_{16}}T_{v_9}) = ((v_6 : F : b1)(v_{14} : G : c1)(v_{16} : H : d3)(v_9 : I : e1))$, $F_3[R] = (T_{v_6}T_{v_{17}}T_{v_{18}}T_{v_9}) = ((v_6 : B :$

$b1)(v_{17} : G : c1)(v_{18} : H : d2)(v_9 : I : e1)$). We see that all R-tuples are value distinct.

Now we are ready to define XID satisfaction.

Definition 2 (XML Inclusion Dependency Satisfaction). *An XML document T satisfies an XML inclusion dependency $\Upsilon(Q, (\{P_1, \cdots, P_n\} \subseteq \{R_1, \cdots, R_n\}))$, denoted as $T \prec \Upsilon$ if there exists a P-tuple in T, then there must be a R-tuple where $(T^{P_i}) =_v (T^{R_i})$ and $i \in [1, \cdots, n]$.*

Now consider the XID $\Upsilon(A, (\{U/B, U/M/C, U/M/D, U/E\} \subseteq \{W/F, W/N/G, W/N/H, W/I\}))$ on the DTD D in Fig. 1(a). We have already showed how to produce tuples for paths in P and R. We see that for all P-tuples, there exists a R-tuple. Thus the XID Υ is satisfied by the document T in Fig. 1(b).

Now we define XML foreign key(XFK).

Definition 3 (XML Foreign Key). *Given an XID $\Upsilon(Q, (\{P_1, \cdots, P_n\} \subseteq \{R_1, \cdots, R_n\}))$ on the DTD, we define XFK as $\digamma(Q, (\{P_1, \cdots, P_n\} \subseteq \{R_1, \cdots, R_n\}))$ if there is an XML key as $\Bbbk(Q, \{R_1, \cdots, R_n\})$.*

For example, we define an XFK as $\digamma(A, (\{U/B, U/M/C, U/M/D, U/E\} \subseteq \{W/F, W/N/G, W/N/H, W/I\}))$ on the DTD D in Fig. 1(a).

Before defining XFK satisfaction, we briefly describe the XML key satisfaction [14]. An XML key $\Bbbk(Q, \{R_1, \cdots, R_n\})$ is satisfied by the document T if all the R-tuples are value distinct in T. For example, we see that the XML key $\Bbbk(A, \{W/F, W/N/G, W/N/H, W/I\})$ on the DTD D in Fig. 1(a) is satisfied by the document T in Fig. 1(b) because all R-tuples are value distinct in the document T.

Definition 4 (XML Foreign Key Satisfaction). *An XML document T satisfies the XFK $\digamma(Q, (\{P_1, \cdots, P_n\} \subseteq \{R_1, \cdots, R_n\}))$ denoted as $T \prec \digamma$ if both XID $\Upsilon(Q, (\{P_1, \cdots, P_n\} \subseteq \{R_1, \cdots, R_n\}))$ and XML key $\Bbbk(Q, \{R_1, \cdots, R_n\})$ are satisfied by the document T.*

We see that the XFK $\digamma(A, (\{U/B, U/M/C, U/M/D, U/E\} \subseteq \{W/F, W/N/G, W/N/H, W/I\}))$ is satisfied by the document T because both XID $\Upsilon(A, (\{U/B, U/M/C, U/M/D, U/E\} \subseteq \{W/F, W/N/G, W/N/H, W/I\}))$ and the XML key $\Bbbk(A, \{W/F, W/N/G, W/N/H, W/I\})$ are satisfied by the document T.

We have just finished the definitions for XID and XFK. Now we give the experimental results on checking both XID and XFK satisfactions. All experiments are implemented in Java using a PC with Intel(R) Centrino Duo CPU T2050 at 1.60GHz, 1.49GB RAM and Microsoft Windows XP.

3 Checking XID Satisfactions

As mentioned before, there are two major tasks in checking XID satisfactions: (a) generation P- tuples and R-tuples and (b) checking the inclusion of P-tuples in R-tuples. First, we present the algorithms of generation of tuples.

In tuple generation, we accomplish two tasks: parsing the document and pairing the values of elements to produce tuples while parsing. Here the term pairing means the process of computing the product of relevant hedges. For example, if the relevant hedges are $H_a = T_1T_2$, $H_b = T_3T_4$ and $H_c = T_5T_6$, pairing produces the tuples (T_1, T_3, T_5), (T_1, T_3, T_6), (T_1, T_4, T_5), (T_1, T_4, T_6), (T_2, T_3, T_5), (T_2, T_3, T_6), (T_2, T_4, T_5), and (T_2, T_4, T_6). Product calculation itself is not difficult, but in the process of pairing, product calculation has to be combined with parsing.

The subsection presents two algorithms. The algorithm 1 shows the parsing and the algorithm 2 shows the pairing and tuple generation. In parsing, we first find the nodes QN for the selector path Q. We then proceed to find the occurrences of elements for paths dependent(P paths) in order of DTD under a selector node. Note that paths in dependent of an XID can appear as a set. But we order the paths of dependent of an XID according to the order of the elements of DTD that involve XID(we omit this process from the algorithm1 for simplicity). We keep track of the occurrences of the elements which are the last elements of dependent paths so that the pairings can be done to produce P-tuples.

In the same way, we produce the R-tuples for paths referenced of an XID.

Data: An XML document T, An XID $\Upsilon(Q, (\{P_1, \cdots, P_n\} \subseteq \{R_1, \cdots, R_n\}))$
Result: A set of P-tuples,$F[P] = (T^{P_1} \cdots T^{P_n})$ and
A set of R-tuples,$F[R] = (T^{R_1} \cdots T^{R_n})$
Let QN= all Q nodes in T
foreach node in QN do
 | **FIND_TUPLES**($\{P_1, \cdots, P_n\}$,node);
end
foreach node in QN do
 | **FIND_TUPLES**($\{R_1, \cdots, R_n\}$,node);
end
//procedure
FIND_TUPLES($\{X_1, \cdots, X_n\}$,**node)**
{
if (all of $X_1, \cdots, X_n$ **occurred with order and any pairing before) then**
 | **MAKE_TUPLES**($array\_X_1[], \cdots, (pair_{rs}[]), \cdots, array\_X_n[]$);
end
foreach j=1 to n do
 | **foreach k=j to n-1 do**
 | | Check any need to pair as **PAIRING**($array\_X_j[], array\_X_{k+1}[]$);
 | **end**
end
Store value for path X_i to $array\_X_i[]$ according to the order of the fields and keep track of order;
}

Algorithm 1. Parsing the document

```
//procedure
PAIRING(array_X_r[], array_X_s[])
{
foreach i=1 to sizeof(array_X_r[]) do
    foreach j=1 to sizeof(array_X_{r+1}[]) do
        ...
        foreach k=1 to sizeof(array_X_s[]) do
            pair_rs[][1]=array_X_r[i];
            pair_rs[][2]=array_X_{r+1}[j];
            ...
            pair_rs[][s-r]=array_X_s[k];
        end
        ...
    end
end
}
//procedure
MAKE_TUPLES(array_X_1[], ..., (pair_rs[]), ..., array_X_n[])
{
foreach i=1 to sizeof(array_X_1[]) do
    foreach j=1 to sizeof(pair_rs[][]) do
        foreach k=1 to sizeof(array_X_n[]) do
            tuple_{1...n}[][1] = array_X_1[i];
            ...
            tuple_{1...n}[][r] = pair_rs[j][1];
            ...
            tuple_{1...n}[][s] = pair_rs[j][s-r];
            ...
            tuple_{1...n}[][n] = array_X_n[k];
        end
    end
end
}
```

Algorithm 2. Pairing of values for fields and generation of tuples

We give an example to illustrate how to produce P-tuples and R-tuples using the above algorithms. Consider the XID $\Upsilon(A, (\{U/B, U/M/C, U/M/D, U/E\} \subseteq \{W/F, W/N/G, W/N/H, W/I\}))$ on the DTD D in Fig. 1(a). Here, the selector path is A, the paths are $U/B, U/M/C, U/M/D, U/E$ are dependent paths(P-paths) and $W/F, W/N/G, W/N/H, W/I$ are referenced paths(R-paths).

In algorithm 1, we take the document T and the XID Υ as inputs. First, we get the Q node v_r. We then take the P-paths to produce P-tuples for node v_r. We get the value $b1$ for node v_3 using path U/B, value $c1$ for node v_{10} using path $U/M/C$, values $d1, d1, d2$ for nodes v_{11}, v_{12}, v_{13} using path $U/M/D$ and value $e1$ for node v_5 using path U/E. There is no need to call PAIRING procedure as there is no repeating occurrences of a set of elements in the document. So we call the procedure MAKE_TUPLES to produce P-tuples. So we get the P-tuples $F_1[P] = ((b1)(c1)(d1)(e1))$, $F_2[P] = ((b1)(c1)(d1)(e1))$ and $F_3[P] = ((b1)(c1)(d2)(e1))$.

We omit the node identifier v and element label e from the tuples because we consider the value comparison in satisfactions.

Now we show how to produce R-tuples for paths $W/F, W/N/G, W/N/H, W/I$. For the selector node v_r, the value for node v_6 using path W/F is $b1$. Then we get the value $c1$ for node v_{14} using path $W/N/G$ and the values $d1, d3$ for nodes v_{15}, v_{16} using path $W/N/H$. We again see the repeating occurrence of values $c1$ for node v_{17} using path $W/N/G$ and the value $d2$ for node v_{18} using path $W/N/H$. Thus we call the procedure PAIRING for the values for the elements G, H. So we get the pair-wise values $(c1, d1), (c1, d3)$ and $(c1, d2)$. At last, we call the procedure MAKE_TUPLES with values of F element, pair-wise (G, H) values and I value. So we get the R-tuples $F_1[R] = ((b1)(c1)(d1)(e1))$, $F_2[R] = ((b1)(c1)(d3)(e1))$ and $F_3[R] = ((b1)(c1)(d2)(e1))$.

We have shown how to generate P-tuples and R-tuples. We are now ready to check the satisfactions of XID. In checking satisfactions of XID, we use hashing technique using Java HashTable. After generating tuples, we put the values of R-tuples as key[1] in the hashtable. We then take each P-tuple and check the hashtable to see whether there exists a R-tuple that is value equivalent to that P-tuple. If there exists a R-tuple for each P-tuple in the hashtable, we say that the XID is satisfied.

We now present the experimental results of checking XID satisfactions. We take the DTD D in Fig. 1(a) and we generate XML documents with different structures and different sizes. By different structures, we mean *repeating* structure and $non - repeating$ structure. By repeating structure, we mean the multiple occurrences of elements in the hedge and by non-repeating structure, we mean the single occurrence of the elements in the hedge in the document. With single occurrence, an element or a group of elements appears only once while with multiple occurrence, an element or a group of elements appear more than one time in a hedge.

In the case of repeating structure, elements need to be combined from different occurrences to form a tuple. For the same number of tuples, if the structure is non-repeating, we need larger number of elements in the document, which means larger document size. In contrast, if the structure is repeating, because of production, a small document will make the number of tuples.

We first study the experimental results of XID satisfactions for the documents with non-repeating structure in Fig. 2 and Fig. 3. For non-repeating structure, the complexity of tuple generation is $O(2n|e|)$ where n is the number of fields and $|e|$ is the average number of occurrences of an element in a hedge. The cost of parsing is $n|e|$. As we use breadth first search in traversing the document, thus we need to traverse $n|e|$ element nodes. In this case, we assume all element nodes are at the same level under a parent(context) node. The cost of pairing is $n|e|$ for close elements in the hedge. This is because in each pair, there is only one occurrence of each element. We note here that the complexity of hashing is nearly constant and linear but it is slightly increasing in most checking. However,

[1] This key is in Java Hashtable(Key,Value), not key constraints in XML.

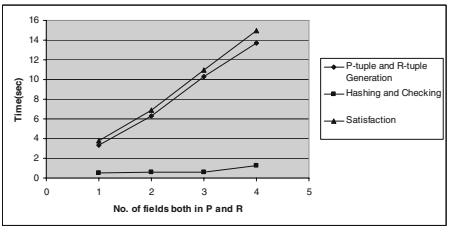

Fig. 2. XID satisfaction time when the number of tuples is fixed to 300K for P and 300K for R

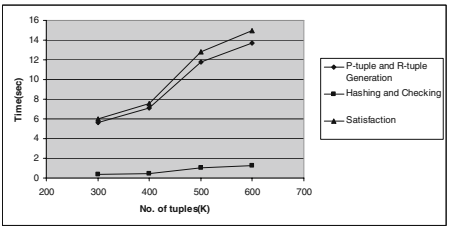

Fig. 3. XID satisfaction time when the number of paths is fixed to 4 in both P and R

sometimes, the time in hashing is increased in non-linear fashion due to the Java hashtable management.

In Fig. 2, the XID satisfaction time is shown where the number of tuples is fixed to 300K both for P-tuples and R-tuples. Thus the total number of tuples in the document is 600K. The satisfaction time for XID is the sum of the tuple generation time and the hashing time of R-tuples and then the inclusion checking time of P-tuples in R-tuples. Both tuple generation time and hashing time with inclusion checking time are linear, but the tuple generation time is increasing because the number of paths is increased in XID whereas the the hashing with inclusion checking time is slightly increasing and this is due to Java Hashtable management to maintain the values for increasing number of paths. We see that tuple generation time much higher than the hashing with inclusion checking time.

We show the XID satisfaction time where we fix the number of paths to 4 both in P and R in Fig. 3. Here, the XID satisfaction time is also linear because the tuple generation time and the hashing and the inclusion checking time are also linear. The tuple generation time is increasing because the number of tuples to be produced is increasing. The hashing and the inclusion checking is slightly increasing due to Java Hashtable management to maintain values of the increasing number of tuples.

We now study the experimental results of XID satisfactions for documents with repeating structure in Fig. 4 and Fig. 5. For the repeating structure in the document, the complexity of tuple generation is $O(n\sqrt[n]{|e|} + |e|)$ where n is the number of fields and $|e|$ is the average number of occurrences of an element in a hedge. The cost $n\sqrt[n]{|e|}$ is for parsing using breadth first search. The cost $|e|$ is for pairing because we do the production of elements.

In Fig. 4, the XID satisfaction time is shown where we fix the number of P-tuples to 300K and R-tuples to 300K and we vary the number of paths in P and R. We start with the number of paths two because the pairing has to be done with at least two paths. Here, the P-tuple and R-tuple generation time is smaller than the hashing time and the inclusion checking time. This is because the parsing time is smaller for a document with repeating structure than that of a document with non-repeating structure. However, the satisfaction time is

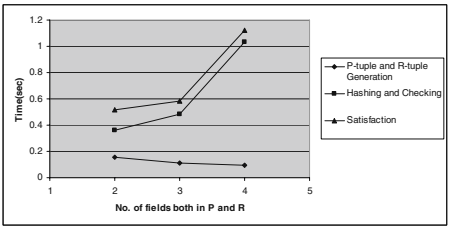

Fig. 4. XID satisfaction time when the number of tuples is fixed to 300K for P and 300K for R

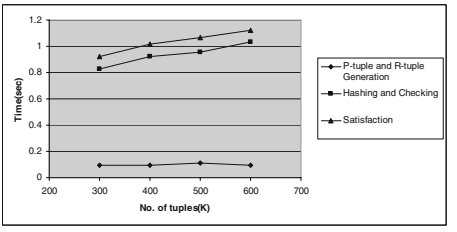

Fig. 5. XID satisfaction time when the number of paths is fixed to 4 in both P and R

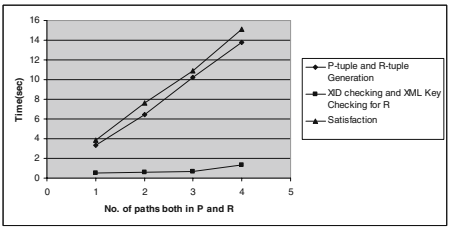

Fig. 6. XFK satisfaction time when the number of tuples is fixed to 300K for P and 300K for R

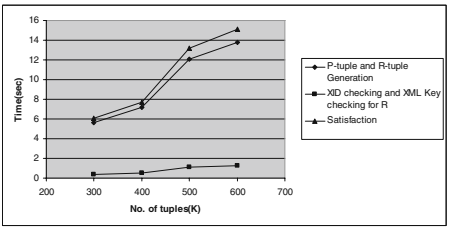

Fig. 7. XFK satisfaction time when the number of paths is fixed to 4 in both P and R

linear considering the tuple generation time and the hashing and the inclusion checking time are linear. We also see that the hashing time is increasing while the tuple generation time is decreasing linearly.

We show the XID satisfaction time in Fig. 5 where we fix the number of paths to 4 but we vary the number of tuples. Like Fig. 4, the XID satisfaction time is linear with the increasing number of tuples to be checked. The tuple generation time is significantly smaller that the hashing and the inclusion checking time.

4 Checking XFK Satisfactions

In this section we study the performance of checking XML foreign key satisfaction. In checking XFK satisfaction, we need two checking: checking inclusion of P-tuples in R-tuples(XID checking) and whether R-tuples are distinct(XML key checking). We already showed how to check XID in the previous section. We only add the XML key checking for paths R to check the satisfaction of XFK. In checking XFK, we only show the experimental results for non-repeating structure of the document.

In Fig. 6, the XFK satisfaction time is shown where we fix the number of tuples to 300K both for P-tuples and R-tuples but we vary the number of paths in P and R. We see that tuple generation time is linear and it is increasing.

But the XID of P-tuples in R-tuples checking and the XML key checking of R-tuples is much smaller than the tuple generation time, but it is linear and slightly increasing. Thus the XFK satisfaction time is linear with the increasing number of paths in both P and R.

We show the XFK satisfaction time in Fig. 7 where we fix the number of paths to 4 but we vary the number of tuples. The XFK satisfaction time is linear and increasing. The tuple generation time is more increasing than the checking of inclusion of P-tuples in R-tuples and the checking of distinctness of R-tuples.

5 Conclusions

We have showed the performances of checking XML inclusion dependency and XML foreign key satisfactions. A novel algorithm for tuple generation is used in checking satisfactions. Both XID and XFK satisfactions can be checked in linear time with the increasing number of tuples in the document and the increasing number of paths in XID and XFK.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Ramakrishnan, R., Gehrke, J.: Database Management Systems. McGraw-Hill Higher Education, New York (2003)
3. Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: Data Integration under Integrity Constraints. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 262–279. Springer, Heidelberg (2002)
4. Bykowski, A., Daurel, T., Meger, N., Rigotti, C.: Integrity Constraints Over Association Rules. In: Meo, R., Lanzi, P.L., Klemettinen, M. (eds.) DSDMA 2004. LNCS (LNAI), vol. 2682, pp. 306–323. Springer, Heidelberg (2004)
5. Fankhouser, P., Klement, T.: XML for Datawarehousing Chances and Challenges. In: Kambayashi, Y., Mohania, M., Wöß, W. (eds.) DaWaK 2003. LNCS, vol. 2737, pp. 1–3. Springer, Heidelberg (2003)
6. Zamboulis, L.: XML Data Integration by Graph Restructuring. In: Williams, H., MacKinnon, L.M. (eds.) BNCOD 2004. LNCS, vol. 3112, pp. 57–71. Springer, Heidelberg (2004)
7. Buneman, P., Fan, W., Simeon, J., Weinstein, S.: Constraints for Semistructured Data and XML. SIGMOD Record, 47–54 (2001)
8. Fan, W.: XML Constraints: Specification, Analysis, and Applications. In: DEXA, pp. 805–809 (2005)
9. Fan, W., Simeon, J.: Integrity constraints for XML. In: PODS, pp. 23–34 (2000)
10. Fan, W., Libkin, L.: On XML Integrity Constraints in the Presence of DTDs. Journal of the ACM 49, 368–406 (2002)
11. Vincent, M.W., Schrefl, M., Liu, J., Liu, C., Dogen, S.: Generalized inclusion dependencies in XML. In: Yu, J.X., Lin, X., Lu, H., Zhang, Y. (eds.) APWeb 2004. LNCS, vol. 3007, pp. 224–233. Springer, Heidelberg (2004)
12. Karlinger, M., Vincent, M., Scherefl, M.: On the Definition and Axiomitization of Inclsuion Dependency for XML, Tecnical Report, No. 07/02, Johanne Kepler University (2007)

13. Shahriar, Md., S., Liu, J.: On Defining Referential Integrity for XML. In: IEEE International Symposium of Computer Science and Its Applications (CSA), pp. 286–291 (2008)
14. Shahriar, Md., S., Liu, J.: On Defining Keys for XML. In: IEEE CIT 2008, Database and Data Mining Workshop, DD 2008, pp. 86–91 (2008)
15. Liu, Y., Yang, D., Tang, S., Wang, T., Gao, J.: Validating key constraints over XML document using XPath and structure checking. Future Generation Computer Systems 21(4), 583–595 (2005)
16. Chen, Y., Davidson, S.B., Zheng, Y.: XKvalidator: a constraint validator for XML. In: CIKM, pp. 446–452 (2002)
17. Buneman, P., Davidson, S., Fan, W., Hara, C., Tang, W.C.: Keys for XML. WWW 10, 201–210 (2001)
18. Vincent, M.W., Liu, J.: Functional dependencies for XML. In: Zhou, X., Zhang, Y., Orlowska, M.E. (eds.) APWeb 2003. LNCS, vol. 2642, pp. 22–34. Springer, Heidelberg (2003)
19. Arenas, M., Libkin, L.: A Normal Form for XML documents. In: ACM PODS, pp. 85–96 (2002)
20. Hartmann, S., Köhler, H., Link, S., Trinh, T., Wang, J.: On the Notion of an XML Key. In: Schewe, K.-D., Thalheim, B. (eds.) SDKB 2008. LNCS, vol. 4925, pp. 103–112. Springer, Heidelberg (2008)
21. Vincent, M.W., Liu, J.: Multivalued Dependencies in XML. In: James, A., Younas, M., Lings, B. (eds.) BNCOD 2003. LNCS, vol. 2712, pp. 4–18. Springer, Heidelberg (2003)
22. Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0., World Wide Web Consortium (W3C) (Febuary 1998), `http://www.w3.org/TR/REC-xml`
23. Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N.: XML Schema Part 1:Structures, W3C Working Draft (April 2000), `http://www.w3.org/TR/xmlschema-1/`
24. Java Document Object Model (DOM), `http://java.sun.com/j2se/1.4.2/docs/api/orgw3c/dom/package-summary.html`