2005

# XTFDs: FDs for XML Documents

Wai Yin Mok
*University of Alabama - Huntsville*, mokw@uah.edu

# XTFDs: FDs for XML Documents

**Wai Yin Mok, Ph.D.**[1]
University of Alabama in Huntsville
mokw@uah.edu

## ABSTRACT

Functional Dependencies (FDs) are a common constraint for many applications. Specifying FDs in XML documents, however, is difficult because XML documents do not have uniform structures. We introduce XML Template Functional Dependencies (XTFDs), which can specify FDs in XML documents, in this paper. Previously, we defined XTFDs in terms of variables only (Mok, 2005). In this paper, we extend our previous work by incorporating XPath expressions. By incorporating XPath expressions, we can express more FDs in XML documents. Since XTFDs are based on simple concepts like variables, functions, and XPath expressions, XTFDs are more intuitive than other proposals of the same purpose in the literature. We are currently comparing XTFDs with the approaches in Arenas & Libkin (2004); Vincent et al. (2004). A preliminary comparison shows that XTFDs improve the approaches in Arenas & Libkin (2004); Vincent et al. (2004) if recursive structures, mixed content and optional attributes are allowed.

### Keywords

XML Template Functional Dependencies, Functional Dependencies, Template Dependencies, XPath expressions.

## INTRODUCTION

This paper introduces XML Template Functional Dependencies (XTFDs)—a new constraint for XML documents. XTFDs are inspired by Template Dependencies (TDs) (Sadri & Ullman, 1982) and XPath expressions (W3C, 1999). To provide a brief overview, XTFDs are constructed from variables and XPath expressions and the semantics of XTFDs are based on predicate calculus. Since XTFDs are built on simple concepts like variables, functions, and XPath expressions,XTFDs are a highly intuitive constraint. Here, we first point out the notation and the basic assumptions of this paper. Element names, element instances, attribute names, and text strings all appear in `typewriter` style. `Person`, `<Person Name="Esau">`, `Name`, `"Jacob"` and `"A plain man"` in Figure 2a are examples. As in Arenas & Libkin (2004); Fan & Libkin (2002); Vincent et al. (2004), we assume every element instance in an XML document is distinct. For example, all `Bar` elements in Figure 3 are unique. Even though `<Bar C="4">text4</Bar>` and `<Bar C="4">text4</Bar>` appear to be the same in Figure 3, they are actually different `Bar` elements which happen to contain the same text string `"text4"` and the same attribute value `"4"`. A text string, on the other hand, is identified by its value. Thus, all occurrences of a text string in an XML document are considered to be identical. Although the text string `"1"` (which is also an attribute value), appear several times in Figure 3, they are all identical. In a sense, we can think of elements as containers that contain other data values such as text strings, numbers and attribute values. Hence, it is possible that two different containers contain the same data values.

## TDS AND XPATH EXPRESSIONS

To demonstrate some sample TDs, let R = ABC be a relation scheme. Figure 1(a) shows a TD that denotes the multivalued dependency $A \rightarrow\rightarrow B \mid C$. The two rows above the horizontal line constitute the hypothesis of the TD and the row below the horizontal line is the conclusion. Figure 1(a) means for any relation r over R, if there is a function φ such that φ maps each $a_i$ to an A-value in r, each $b_i$ to an B-value in r, each $c_i$ to an C-value in r, and $(\varphi(a_1), \varphi(b_1), \varphi(c_1)) \in r$ and $(\varphi(a_1), \varphi(b_2), \varphi(c_2)) \in r$, then $(\varphi(a_1), \varphi(b_1), \varphi(c_2)) \in r$ as well. Figure 1(b) shows the FD $A \rightarrow B$. The meaning of its hypothesis is the same as that of $A \rightarrow\rightarrow B \mid C$. Its conclusion, however, means $\varphi(b_1) = \varphi(b_2)$.

We assume the reader is familiar with basic XPath expressions up to the level of Carey (2004). Here, we demonstrate some XPath expressions that are particularly useful for this research. As an example, we show an incomplete Abraham's family tree in Figure 2(a), an XSLT style sheet that uses several XPath expressions in Figure 2(b), and the result document of

[1] W.Y. Mok was supported in part by the Richard A. Witmondt Faculty Fellowship and a UAH Research Mini-Grant.

$a_1\ b_1\ c_1$
$a_1\ b_2\ c_2$
---
$a_1\ b_1\ c_2$

(a)

$a_1\ b_1\ c_1$
$a_1\ b_2\ c_2$
---
$b_1 = b_2$

(b)

Figure 1: Sample TDs

applying this style sheet in Figure 2(c). We have validated Figures 2(a) and 3 by using the services provided at http://www.stg.brown.edu/service/xmlvalid/ and produced Figure 2(c) by using Microsoft Internet Explorer 6.0.

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl"
                 href="person.xsl" ?>

<!DOCTYPE BiblicalFigures
[
<!ELEMENT BiblicalFigures (#PCDATA | Person)*>
<!ELEMENT Person (#PCDATA | Person)*>
<!ATTLIST Person Name CDATA #REQUIRED>
]>

<BiblicalFigures>
  A father of many nations<Person Name="Abraham">
    The son of Hagar<Person Name="Ishmael"/>
    Laid on the altar<Person Name="Isaac">
      A cunning hunter<Person Name="Esau">
        The son of Adah<Person Name="Eliphaz"/>
      </Person> <!-- End of Esau's line -->
      A plain man<Person Name="Jacob">
        Jacob's firstborn<Person Name="Reuben"/>
        A lion's whelp<Person Name="Judah"/>
        A fruitful bough<Person Name="Joseph"/>
      </Person> <!-- End of Jacob's line -->
    </Person>
  </Person>
</BiblicalFigures>
```

(a)

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">

  Example 1:<br/>
  <xsl:for-each select=
    "/BiblicalFigures/Person/Person/Person/@Name">
    <xsl:value-of select="." /><br/>
  </xsl:for-each>

  Example 2:<br/>
  <xsl:for-each select=
    "//Person/Person/Person/@Name">
    <xsl:value-of select="." /><br/>
  </xsl:for-each>

  Example 3:<br/>
  <xsl:for-each select=
    "//text()[2]">
    <xsl:value-of select="." /><br/>
  </xsl:for-each>

  Example 4:<br/>
  <xsl:for-each select=
    "//Person[3]/@Name">
    <xsl:value-of select="." /><br/>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

(b)

```
Example 1:
Esau
Jacob
Example 2:
Esau
Eliphaz
Jacob
Reuben
Judah
Joseph
Example 3:
Laid on the altar
A plain man
A lion's whelp
Example 4:
Joseph
```

(c)

Figure 2: An incomplete Abraham's family tree

Example 1 in Figure 2b demonstrates an absolute path—an XPath expression that begins with the symbol /. Since it is an absolute path, `BiblicalFigures` in this expression must map to the root element of the XML document in Figure 2a. The first `Person` in the expression can only map to `<Person Name="Abraham">`, a child `Person` element of the root element. The second `Person` in the expression can only map to `<Person Name="Isaac">`, a child `Person` element of `<Person Name="Abraham">`. We have a choice for the third `Person` in the expression, however. It can map to `<Person Name="Esau">` or `<Person Name="Jacob">`, both are child `Person` elements of `<Person Name="Isaac">`. Thus, `@Name` in the expression can map to `"Esau"` or `"Jacob"`, as shown in Figure 2c. Example 2 in Figure 2b shows an XPath expression that begins with the symbol //, which means it is not an absolute path. Now the first `Person` in this expression can map to `<Person Name="Abraham">` or `<Person Name="Isaac">`. This results in six names, as shown in Figure 2c. Example 3 uses the expression `//text()[2]`, which can map to the second text string of any element. The result is the three text strings in Figure 2c. Example 4 uses the expression `//Person[3]/@Name`. `Person[3]` can map to the third child `Person` element of any element. Hence, the entire expression can map to the value of the `Name` attribute of the third child `Person` element of any element. This time only `"Joseph"` satisfies.

### XTFDS

We emphasize that the definition of XTFDs does not depend on the definition of DTDs. In fact, we can use XML schema (W3C, 2004b) in this paper rather than DTDs and the results will remain the same. Given a DTD **D**, let **E** be the set of element names of **D** and let **A** be the set of attribute names of **D**. To define an XTFD with respect to **D**, we first derive several sets of variables from **E** and **A**. In the following, $\mathbf{Z}^+ = \{1, 2, 3, \ldots\}$.

- For each $e \in \mathbf{E}$, $var(e) = \{e_i \mid i \in \mathbf{Z}^+\} \cup \{e[k]_i \mid i, k \in \mathbf{Z}^+\}$.
- For each $a \in \mathbf{A}$, $var(a) = \{a_i \mid i \in \mathbf{Z}^+\}$.
- **strvar** $= \{text()_i \mid i \in \mathbf{Z}^+\} \cup \{text()[k]_i \mid i, k \in \mathbf{Z}^+\}$.

For example, $Person_2$, $Person[3]_5 \in var(Person)$, $Name_7 \in var(Name)$, and $text()_{11}$, $text()[13]_{17} \in$ **strvar**. We further let $var(\mathbf{E})$ be $\bigcup_{(e \in \mathbf{E})} var(e)$ and let $var(\mathbf{A})$ be $\bigcup_{(a \in \mathbf{A})} var(a)$. Syntactically, an XTFD consists of a hypothesis, which is a (potentially empty) finite set of XTFD-elements, and a conclusion. Each XTFD-element has one of the following forms:

- $\alpha\beta_1/\beta_2/\ldots/\beta_m$, $m \geq 1$,
- $\alpha\beta_1/\beta_2/\ldots/\beta_m/\gamma$, $m \geq 1$,
- $//\gamma$.

The meanings of the symbols $\alpha$, $\beta$'s, and $\gamma$ are as follows:

- $\alpha = /$ or $\alpha = //$,
- $\beta_i \in var(\mathbf{E})$, $1 \leq i \leq m$,
- $\gamma = @a_i$ where $a_i \in var(\mathbf{A})$ or $\gamma \in$ **strvar**.

The conclusion of an XTFD has the following form:

- $\chi_1 = \chi_2$ where $\chi_1$ and $\chi_2$ are both XTFD-elements.

Like Sadri & Ullman (1982), we write each XTFD-element of the hypothesis of an XTFD above a horizontal line and the conclusion below the horizontal line. The XTFDs in Figure 4 are all syntactically correct with respect to Figure 3.

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl"
                 href="redundancy.xsl" ?>

<!DOCTYPE Foo
[
<!ELEMENT Foo ANY>
<!ELEMENT Bar ANY>
<!ATTLIST Foo A CDATA #IMPLIED
              B CDATA #IMPLIED>
<!ATTLIST Bar C CDATA #IMPLIED>
]>
```

```
<Foo A="1" B="2">
  text1
  <Foo A="1" B="3"/>
  text2
  <Bar>
    <Foo A="1" B="2"/>
    <Bar/>
    <Foo A="1"/>
  </Bar>
  text3
  <Bar C="4">text4</Bar>
  <Bar C="4">text4</Bar>
</Foo>
```

Figure 3: An XML document with redundant data

```
                        //Bar1/@C1
                        //Bar2/@C1
//Foo1/@A1              //Bar1/text()[1]1          /Foo1/Bar1/Foo2/@A1
//Foo2/@A1              //Bar2/text()[1]2          /Foo3/Bar2/Foo4/@A1
----------------------  --------------------------  ------------------------------------
//Foo1/@B1 = //Foo2/@B2  //text()[1]1 = //text()[1]2  /Foo1/Bar1/Foo2/@B1 = /Foo3/Bar2/Foo4/@B2
```

$XTFD_1$ $XTFD_2$ $XTFD_3$

```
------------
//@B1 = //@B2
```

$XTFD_4$

Figure 4: Sample XTFDs

We now turn our attention to the semantics of an XTFD. Let **E** and **A** be the sets of element names and attribute names of a given DTD **D** respectively and let **Doc** be a valid XML document with respect to **D**. We define the following sets with respect to **Doc** as follows:

- For each $e_i \in$ var(**E**), ext($e_i$) = {ele | ele is an e element in **Doc**}.
- For each $e[k]_i \in$ var(**E**), ext($e[k]_i$) = {ele | ele is the kth e element of an element in **Doc** or of /}.
- For each $a_i \in$ var(**A**), ext($@a_i$) = {val | val is the value of the a attribute of an element in **Doc**}.
- For each $text()_i \in$ **strvar**, ext($text()_i$) = {val | val is a text string of an element in **Doc**}.
- For each $text()[k]_i \in$ **strvar**, ext($text()[k]_i$) = {val | val is the kth text string of an element in **Doc**}.

Let r be the root element in **Doc**, and let **T** be the node tree model of **Doc**. Note that / in the membership condition of ext($e[k]_i$) is the root node in **T** (Carey, 2004). No element except r in **Doc**, no attribute in **Doc** and no text string in **Doc** is a child of /, although they are all descendants of /. For example, <Foo A="1" B="2">—the root element of the XML document in Figure 3—is a child element of /. Thus, <Foo A="1" B="2"> ∈ ext($Foo[1]_7$). Using Figure 3 as an example, ext($Foo_1$) is the set of all four Foo elements, ext($Foo[2]_4$) = {<Foo A="1"/>}, ext($@A_9$) = {"1"}, ext($@B_{16}$) = {"2", "3"}, ext($text()_{25}$) = {"text1", "text2", "text3", "text4"} and ext($text()[2]_{36}$) = {"text2"}.

Like TDs, we use functions to relate XTFD-elements and valid XML documents. Specifically, an XTFD-element $\chi$ is mappable to a valid XML document **Doc** if there is a function $\varphi$ that satisfies all of the following conditions:

- $\varphi(v) \in$ ext(v) if $v \in$ var(**E**) $\cup$ **strvar**. $\varphi(@v) \in$ ext(@v) if $v \in$ var(**A**).
- If $\chi$ is of the form $\alpha\beta_1/\beta_2/\ldots/\beta_m$ or of the form $\alpha\beta_1/\beta_2/\ldots/\beta_m/\gamma$, then $\varphi(\beta_i)$ is a child element of $\varphi(\beta_{i+1})$ in **Doc**, $1 \leq i < m$. Additionally, if $\alpha$ = /, then $\varphi(\beta_1)$ is the root element of **Doc**.
- If $\chi$ is of the form $\alpha\beta_1/\beta_2/\ldots/\beta_m/\gamma$, we have three cases for $\gamma$:
    1. If $\gamma = @a_i$, $\varphi(\gamma)$ is the value of the a attribute of $\varphi(\beta_m)$.

2. If $\gamma$ = text()$_i$, $\varphi(\gamma)$ is a text string of $\varphi(\beta_m)$.
3. If $\gamma$ = text()[k]$_i$, $\varphi(\gamma)$ is the kth text string of $\varphi(\beta_m)$.

For the conclusion of an XTFD, if $\chi$ is a mappable XTFD-element under a function $\varphi$, we define $\varphi(\chi)$ as follows:

- $\varphi(\chi) = \varphi(\beta_m)$ if $\chi$ is of the form $\alpha\beta_1/\beta_2/\ldots/\beta_m$.
- $\varphi(\chi) = \varphi(\gamma)$ if $\chi$ is of the form $\alpha\beta_1/\beta_2/\ldots/\beta_m/\gamma$ or of the form $//\gamma$.

Every XTFD-element of every XTFD in Figure 4 is mappable to the XML document in Figure 3. In fact, if we ignore the subscripts, every one of them is a valid XPath expression. However, /Foo$_1$/Bar$_1$/Foo$_1$ is unmappable because $\varphi$(Foo$_1$) cannot be both parent and child elements of $\varphi$(Bar$_1$). A set of XTFD-elements is mappable to a valid XML document if there is a function $\varphi$ under which each member thereof is mappable. Clearly if a set of XTFD-elements is mappable, then each member is mappable separately. The converse of this statement, however, is not true. For example, /Foo$_1$/Bar$_1$/Bar$_2$ and //Bar$_1$/text()$_1$ are each mappable to the XML document in Figure 3. Nevertheless, they are not mappable under a single function.

Let **E** and **A** be the sets of element names and attribute names of a given DTD **D** respectively and let **Doc** be a valid XML document with respect to **D**. **Doc** violates an XTFD where $\{\chi_1, \chi_2, \ldots, \chi_n\}$ (n >= 0) is its hypothesis and $\chi' = \chi''$ is its conclusion if there is a function $\varphi$ such that $\chi_1, \chi_2, \ldots, \chi_n, \chi', \chi''$ are all mappable to **Doc** under $\varphi$ and $\varphi(\chi') \neq \varphi(\chi'')$. On the other hand, if there is not such a function, then **Doc** satisfies the XTFD. For example, the XML document in Figure 3 satisfies XTFD$_2$ and XTFD$_3$ in Figure 4. However, it violates XTFD$_1$ because we can map Foo$_1$ to <Foo A="1" B="2">, Foo$_2$ to <Foo A="1" B="3"/>, and these two Foo elements have the same A attribute value but different B attribute values. Likewise, it also violates XTFD$_4$ because we can map @B$_1$ to "2" and @B$_2$ to "3".

It would be illustrative to explain XTFD$_1$ here because the "trick" we used in XTFD$_1$ is used again in the other XTFDs. We have two Foo element variables in XTFD$_1$, namely Foo$_1$ and Foo$_2$. We can map Foo$_1$ and Foo$_2$ to two (not necessarily distinct) Foo elements in a valid XML document. However, since the attribute variable @A$_1$ appears as the last variable of the first two XTFD-elements, we have to map Foo$_1$ and Foo$_2$ to two Foo elements that have the same A-attribute value. The conclusion of XTFD$_1$ states that the B-attribute value of the Foo element where Foo$_1$ is mapped to and the B-attribute value of the Foo element where Foo$_2$ is mapped to must be the same. In plain English, it means that for any two Foo elements, if they have the same A-attribute value, they must have the same B-attribute value.

## CONCLUSIONS

This paper presents XML Template Functional Dependencies (XTFDs) as a new way to specify functional dependencies in XML documents. XTFDs combine the concepts of XPath expressions and template dependencies. Even though we do not have space to present a preliminary comparison of XTFDs with the approaches in Arenas & Libkin (2004); Vincent et al. (2004) in this paper, we have found that XTFDs improve the approaches in Arenas & Libkin (2004); Vincent et al. (2004) if recursive structures, mixed content and optional attributes are allowed.

## REFERENCES


1. Arenas, M. & Libkin, L. (2004). A normal form for xml documents. *ACM Transactions on Database Systems*, 29 (1), 195-232.
2. Carey, P. (2004). New Perspectives on XML, Comprehensive. Boston, Massachusetts: Course Technology.
3. Fan, W. & Libkin, L. (2002). On xml integrity constraints in the presence of DTDs. *Journal of the ACM*, 49 (3), 368-406.
4. Mok, W. Y. (2005). On Utilizing Variables for Specifying FDs in XML Documents. *submitted for publication*.
5. Sadri, F. & Ullman, J. D. (1982). Template dependencies: A large class of dependencies in relational databases and its complete axiomatization. *Journal of the ACM*, 29 (2),363-372.
6. Vincent, M. W., Liu, J., & Liu, C. (2004). Strong functional dependencies and their application to normal forms in xml. *ACM Transactions on Database Systems*, 29 (3), 445-462.
7. W3C (1999). Xml path language (xpath). http://www.w3.org/TR/xpath.
8. W3C (2004a). Extensible markup language (xml) 1.0 (third edition). http://www.w3.org/TR/REC-xml/.
9. W3C (2004b). Xml schema part 0: Primer second edition. http://www.w3.org/TR/xmlschema-0/.