

# Characterization of the Interaction of XML Functional Dependencies with DTDs

Łucja Kot and Walker White

Department of Computer Science  
Cornell University, Ithaca, NY 14853, USA  
Phone: (607) 255-9537 Fax: (607) 255-4428  
{lucja,wmwhite}@cs.cornell.edu

**Abstract.** With the rise of XML as a standard model of data exchange, XML functional dependencies (XFDs) have become important to areas such as key analysis, document normalization, and data integrity. XFDs are more complicated than relational functional dependencies because the set of XFDs satisfied by an XML document depends not only on the document values, but also the tree structure and corresponding DTD. In particular, constraints imposed by DTDs may alter the implications from a base set of XFDs, and may even be inconsistent with a set of XFDs. In this paper we examine the interaction between XFDs and DTDs. We present a sound and complete axiomatization for XFDs, both alone and in the presence of certain classes of DTDs; we show that these DTD classes induce an axiomatic hierarchy. We also give efficient implication algorithms for those classes of DTDs that do not use disjunction or nesting.

## 1 Introduction

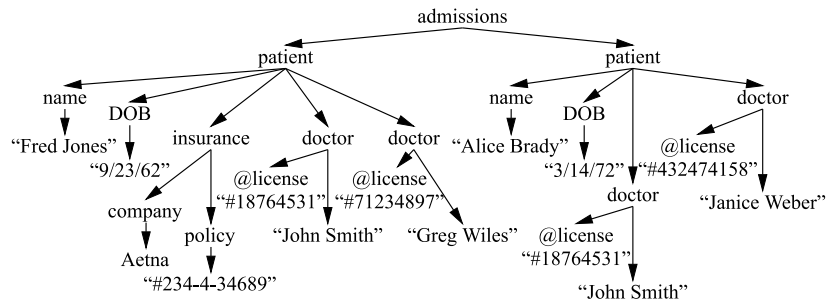
Functional dependencies have proved to be a very useful class of integrity constraints for traditional database systems. They are integral to key analysis, normalization, and query optimization [1]. As XML is increasingly becoming the standard model of data exchange, there is much interest in formulating a definition of XML functional dependency. In addition to the benefits found in relational databases, a proper XFD definition would also aid in many new areas, such as verifying data consistency, preserving semantics during data exchange, and XML-SQL translation [10].

Several different XFD definitions have been suggested [3, 20, 14, 19, 16, 13]. The major XFD definitions are similar to the relational definition except that, instead of attributes and table rows, they use path identifiers and subtrees. Informally, these definitions say that an XFD  $A \rightarrow B$  is satisfied in a document if, for any two subtrees, whenever they agree on the paths in  $A$ , they also agree on the paths in  $B$ . The definitions differ primarily in how they choose subtrees, specify path identifiers, or test equality between XML nodes.

XFDs differ from their relational counterparts in that they must take into account the tree structure of XML documents. For example, a language for defining XFDs must allow us to specify when one path is a prefix of another. Another issue is the definition of equality: nodes can be compared by identity or value equality. In the relational model, no duplicate tuples are allowed, and so value equality is sufficient in FDs. However, in

an XML document, we can have two different subtrees that are isomorphic and have exactly the same values. This is a clear instance of data duplication, but one allowed by the data model. Therefore, if XFDs are to be used for keys or normalization, they must be able to detect identity (in)equality between nodes. Finally, as XML represents semi-structured data that is often incomplete, XFDs must properly handle null values.

The implication problem is fundamental to all of the applications mentioned above [1]; hence this has been the focus of much of the work on XFDs. There are two important approaches to the implication problem. One approach is that of efficient decision algorithms, which allow us to determine whether an XFD is implied by a set of XFDs; some feasible decision algorithms have been discovered already [3]. The other approach is axiomatization, which often gives us slower decision algorithms, but which is important for understanding the underlying theory of XFDs [1]. For example, every child XML node has a unique parent node. Thus for any two path identifiers  $q$  and  $p$ , where  $p$  is an identifier for the parent of  $q$ , every XML document satisfies the XFD  $q \rightarrow p$ . A decision algorithm would allow us to check for *each specific* instance of parent-child  $p, q$  that  $q \rightarrow p$  holds. However, an axiomatization would allow us to prove this entire general class of XFDs.



**Fig. 1.** XML Document for Medical Admissions

The implication problem becomes more complicated in the presence of a DTD. Consider the XML document illustrated in Figure 1. This document represents admissions at a special charity hospital and has the following entry in its DTD:

```
<!ELEMENT patient (name,DOB,insurance?,doctor,doctor)>
```

In particular, each patient must have a recommendation from exactly two doctors (who are unordered), and may or may not have insurance. Note that in this DTD, every patient has exactly one name node. So every document conforming to this DTD must satisfy the XFD  $p \rightarrow q$ , where  $p$  is a path identifier for a patient node, and  $q$  is a path identifier for the name node of that patient. This suggests that we should be able to use the structure of a DTD to make deductions about classes of XFDs satisfied by conforming documents. While there are several decision algorithms for XFDs conforming to certain classes of DTDs, to our knowledge there is no existing sound and complete axiomatization for XFDs with identity equality in the presence of a DTD.

## 1.1 Contributions

This paper is a study of the theory of XFDs and their interaction with various classes of DTDs, focusing on axiomatization and the implication problem. In this paper, we make the following contributions.

- We adapt the definition of XFDs presented in [3] to include documents without a DTD so that we can identify the base theory of XFDs.
- We adapt the chase algorithm to XFDs, and use it to improve existing bounds on the implication problem for XFDs, in some cases to linear time.
- We use the chase to formulate the first sound and complete axiomatization for XFD implication (using identity equality), in the absence of a DTD.
- We expand these techniques to explore the interactions between XFDs and several classes of DTDs. In particular, for all DTDs that can be rewritten without disjunction or nesting, we present an efficient chase algorithm for checking implication, as well as a sound and complete axiomatization of the same problem.

The remainder of the paper is organized as follows: Section 2 introduces the preliminary notation, while Section 3 presents our definition of XFDs. Sections 4 and 5 are devoted, respectively, to the implication problem in the absence and in the presence of a DTD. We discuss related work in Section 6 and conclude in Section 7.

## 2 Preliminaries

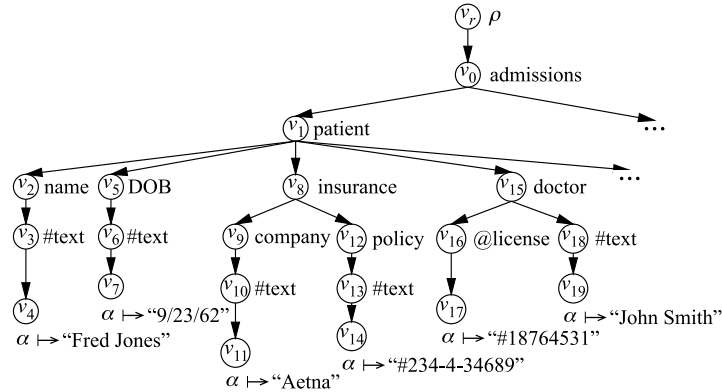
### 2.1 The document model

Throughout this paper, our notation is similar to that in the literature [3], though with some noticeable differences. These differences are necessary because this existing notation requires that an XML document have a corresponding DTD. In order to study the theory of XFDs, we need to decouple the definition of an XFD from a DTD.

Our model for an XML document is a tree representing the underlying DOM structure. We associate each part of the document, including text and attribute data, with a labeled node in the tree. For these labels, we have two disjoint sets  $EL$  and  $VAL$ . The set  $EL$  is the label alphabet for the XML nodes and the attribute names, while  $VAL$  is the alphabet of attribute values.

Formally, our model is the same in as Arenas and Libkin [3] with only two minor modifications. First, our alphabet  $EL$  contains two special elements  $\rho$  and  $\alpha$ . The label  $\rho$  is used to identify the unique root element of each XML tree. This corresponds to the `<?xml>` tag in an XML document, and is necessary because documents without a DTD have no constraints on the root label.

The label  $\alpha$  is used to decouple an attribute from its value. Within any XML document, an attribute is split into two tree nodes: one for its identifier and one for its value. The value node is labeled by  $\alpha$ , and is the sole child of the identifier for that attribute. Furthermore, we have a function that maps each  $\alpha$  node to its value in  $VAL$ . The introduction of  $\alpha$  is a purely technical device; it simplifies the notation in settings where we need to refer to both the address and the value of attribute nodes.



**Fig. 2.** Encoding of Medical XML Document

To illustrate this model, consider the XML document from Figure 1. Our model would represent this document by the labeled tree in Figure 2.

A path identifier is a finite list of labels in EL. For clarity, we separate the elements in a path identifier by periods, such as  $\rho$ .admissions.patient. We say that a path identifier  $p$  occurs in a tree  $T$  if there is a path  $v_1, v_2, \dots, v_n$  in  $T$  such that the labels for the  $v_i$  form a string equal to  $p$ . For the rest of the paper, we refer to path identifiers as *paths*, with the understanding that a single identifier may represent more than one actual path in the tree. We say that a path is *rooted* if its first identifier is  $\rho$ . We denote the set of all rooted paths that occur in  $T$  as  $\text{paths}(T)$ ; note that this is a prefix-closed set.

Finally, we work with two kinds of equality on the nodes of  $T$ : identity equality and value equality. We compare internal tree nodes by identity; on the other hand, we compare  $\alpha$  leaf nodes by their value in VAL.

## 2.2 DTDs

Our definition of a DTD is the standard one [3], namely a pair  $D = (E, P)$ , where  $E$  is a finite subset of EL and  $P$  is a set of productions that map elements of  $E$  to regular expressions over  $E$ . As in the definition in Arenas and Libkin [3], we ignore order in these regular expressions. This is acceptable because XFDs are not influenced by document order.

It is usual notation, when working with DTDs, to use arrows in the notation for productions (i.e.  $a \rightarrow bc^*$ ). In this paper, we need to avoid a notational conflict with another sense of  $\rightarrow$ , which is used for XFDs. Therefore, we use  $\rightsquigarrow$  instead for DTD productions. Informally, a document  $T$  satisfies  $D$ , written  $T \models D$ , if for each node  $v$ , the labels of the children of  $v$  can be produced from the label for  $v$  via  $P$ . A more formal definition is available in Arenas and Libkin [3].

We assume that all DTDs considered from now on are consistent (i.e. there is at least one finite tree satisfying the DTD). Furthermore, we denote by  $\text{paths}(D)$  all the possible rooted paths that may occur in any  $T$  such that  $T \models D$ .

We now define several classes of DTDs, according to the complexity of the regular expressions present in the productions. Bear in mind that we disregard order in defin-

ing these classes. Our DTD classes form a hierarchy and they are described below in increasing order of generality. We note that given a DTD, deciding which class it belongs to may be nontrivial in the worst case. However, we do not expect this to be an issue in practice, as humans do not typically write DTDs with complex nested regular expressions.

**Simple DTDs:** Our definition of a simple DTD is similar to that given in Arenas and Libkin [3]. Given an alphabet  $A$ , a regular expression over  $A$  is called *trivial* if it is of the form  $s_1 \cdots s_n$ , where for each  $s_i$  there is a letter  $a_i \in A$  such that  $s_i$  is either  $a_i, a_i^?, a_i^+$  or  $a_i^*$ , and for  $i \neq j, a_i \neq a_j$ . A simple DTD is one where the right-hand side of any production is trivial. An example of a simple DTD is

$$\rho \rightsquigarrow ab^*, \quad a \rightsquigarrow c^*, \quad b \rightsquigarrow d^+e^*$$

**#-DTDs:** #-DTDs are a proper extension of simple DTDs. This extension allows productions having more than one occurrence of the same alphabet symbol (the # is intended to represent the concept of number). In other words, a #-DTD is a simple DTD which allows the right hand side to contain expressions of the form  $a_i^n$ . This class includes any DTD which does not use disjunction or nesting in its productions. (Nesting refers to parenthesised expressions in productions such as  $\rho \rightsquigarrow (aab)^+$ , which induces a correlation on the numbers of  $a$  and  $b$  children of the root). An example of a non-simple #-DTD is

$$\rho \rightsquigarrow aaab^*, \quad a \rightsquigarrow c^*, \quad b \rightsquigarrow ddeee^*$$

The DTD for the example in Figure 1 is a concrete example of a #-DTD. Every patient must have two recommending doctors. We give no preference to either doctor, and we may want to assert an XFD from the `license` number of a `doctor` to its text content, so we do not wish to give the doctors different tags.

**Arbitrary DTDs:** Arbitrary DTDs represent the most general class of DTDs. They allow all features, including arbitrary disjunction and nesting.

### 2.3 Mapping an XML tree to a nested relation

Many existing definitions of XFDs implicitly rely on the nested relational structure of XML documents. In order to use the existing theory of relational functional dependencies, we make this connection explicit. To each tree  $T$ , we associate a nested relation  $R(T)$ , which we unnest to a flat relation  $U(R(T))$ . Note however, that this translation is conceptual only; we do not normally materialize  $U(R(T))$ .

In this section, we give a high-level illustration of  $U(R(T))$ ; a more formal construction, together with a discussion of the close relationship between  $U(R(T))$  and  $tuples_D(T)$  from Arenas and Libkin [3], appears in the expanded version of this paper [15]. Our illustration makes use of the example in Figure 3. We have separated the tree elements  $v_i$  from their labels in EL to make clear the difference between nodes and their labels.

We start with a tree  $T$  for an XML document. First, we must normalize the tree to make it suitably homogeneous. For each path  $p$  that occurs in  $T$ , we take each prefix  $q$  of  $p$ . For each occurrence of  $q$  in  $T$ , we guarantee that  $q$  can be extended to a path matching  $p$  by adding special null nodes as necessary. In Figure 3,  $\rho.a.b.f.\alpha$  matches a

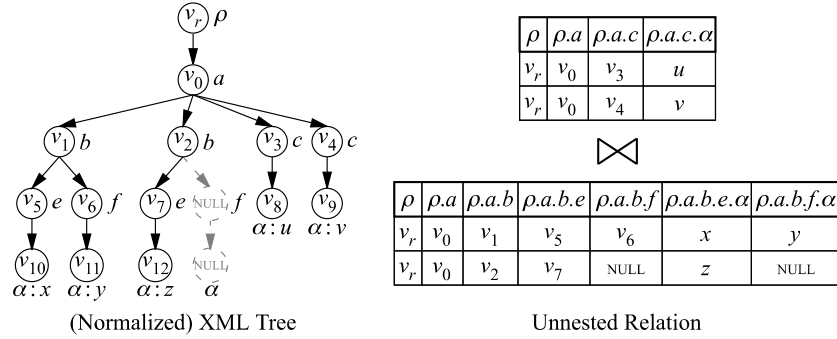


Fig. 3. Mapping an XML Document to an Unnested Relation

path in the tree, so we have to extend the second path matching  $\rho.a.b$  with null nodes to extend it to  $\rho.a.b.f.\alpha$ . If we add only one instance of every required null node, the normalization is unique.

After normalizing the tree, we take each maximal rooted path  $p$  occurring in the tree. We make a table with an attribute for each non-empty prefix of  $p$ . For each match to  $p$  in the normalized tree, we construct a row in this table. In this case, each attribute corresponds to a unique node  $v$  and we assign this attribute  $v$  if  $v$  is an internal node (i.e. the label is not  $\alpha$ ) or is null; otherwise we assign it the attribute value for  $\alpha$  in VAL. The top table on the right hand side of Figure 3 represents the table for the path  $\rho.a.c.\alpha$ .

We then join all of these tables together. For each pair of tables, we use the common prefix of the pair as our join key. The bottom table on the right hand side of Figure 3 is the result of joining the tables for  $\rho.a.b.e.\alpha$  and  $\rho.a.b.f.\alpha$ .

### 3 XML Functional Dependencies

Given the basic notation, we can now define XFDs, and describe the implication and consistency problems. As we mentioned before, our definition of XFDs must account explicitly for the possibility of null values. Several means of handling null values have been suggested for the relational model [4, 17]. We adopt the definition in [4], which is also the one used in Arenas and Libkin [3]: given a relation  $R$  over a set of attributes  $U$  and  $A, B \subseteq U$ ,  $A \rightarrow B$  holds if for any two tuples  $t_1, t_2 \in R$  that agree and are nonnull on all of  $A$ , the  $t_i$  are equal (though possibly both null) for each attribute in  $B$ .

#### 3.1 Tree patterns and functional dependencies

Relational functional dependencies are defined on tuples in the relation. The corresponding notion for an XML tree is a match for a *tree pattern*. Syntactically, a tree pattern  $\varphi$  is a document in which none of the attribute nodes (i.e. nodes labeled  $\alpha$ ) are yet mapped to a value in VAL. A *simple* pattern is a tree pattern where no node has two children with the same label  $l \in \text{EL}$ . In a simple pattern, every rooted path occurs at most once; from now, we assume that all patterns are simple.

In order to use tree patterns to define XML functional dependencies, we must first define what it means to *match* a pattern in a document  $T$ . Intuitively, we want to match the pattern in the document “as far as possible”, allowing incomplete matches only when the document does not allow a full match. Formally, given a tree  $T$  and a pattern  $\varphi$ , a match for  $\varphi$  in  $T$  is a function  $\mu : \text{nodes}(\varphi) \rightarrow \text{nodes}(T) \cup \{\text{null}\}$  where

- $\mu$  maps the root of  $\varphi$  to the root of  $T$ .
- For all nodes  $v$ ,  $\mu(v)$  is either null, or it preserves the label of  $v$ .
- If  $v'$  is a child of  $v$  and  $\mu(v')$  is not null, then  $\mu(v)$  is also not null and  $\mu(v')$  is a child of  $\mu(v)$ .
- If  $v'$  is a child of  $v$  and  $\mu(v')$  is null while  $\mu(v)$  is not, then  $\mu(v)$  could not have had any child with the same label as  $v'$  (i.e. no “premature” null values are allowed).

For any path  $p \in \varphi$ , we use  $\mu(p)$  to represent the image under  $\mu$  for the node at the end of this path. We let  $\mathcal{M}_{\varphi, T} = \{ \mu \mid \mu \text{ is a match for } \varphi \text{ in } T \}$ .

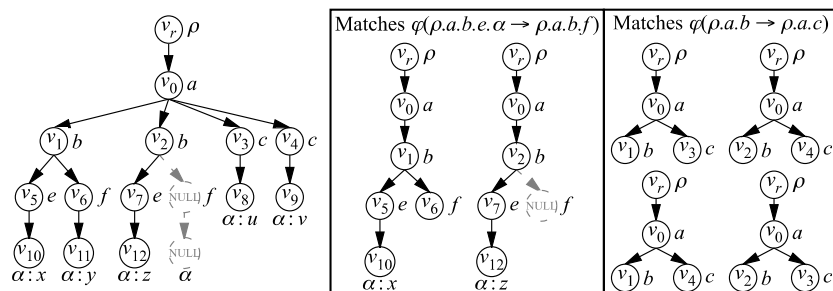


Fig. 4. Sample Tree Pattern Matches

**Definition 1.** A functional dependency  $\sigma = A \rightarrow B$  consists of two subsets  $A, B \subseteq \text{PATHS}$ . We let  $\varphi(\sigma)$  be the smallest tree pattern (with respect to number of nodes) in which all paths in  $A$  and  $B$  occur; note that this pattern is unique. Furthermore, we let  $=_{fd}$  be an equality relation that compares nodes by identity equality if they are not attribute nodes (i.e. labeled by  $\alpha$ ), and by value equality otherwise. The dependency holds if, for any two matches  $\mu_1, \mu_2 \in \mathcal{M}_{\varphi(\sigma), T}$ , whenever we have  $\mu_1(p), \mu_2(p) \neq \text{null}$  and  $\mu_1(p) =_{fd} \mu_2(p)$  for all  $p \in A$ , then for all  $q \in B$ , either  $\mu_1(q) = \mu_2(q) = \text{null}$  or  $\mu_1(q) =_{fd} \mu_2(q)$ .

We illustrate this definition in Figure 4. Recall that our patterns are simple, and so each path in the pattern must occur exactly once. This means that there are only two matches for the pattern  $\varphi(\rho.a.b.e.\alpha \rightarrow \rho.a.b.f)$  within this tree; no match for a simple pattern can contain both  $v_7$  and  $v_6$ . If  $x \neq z$  then the corresponding XFD is satisfied, otherwise it is not, as the  $f$  node is null in one match and non-null in another. Similarly, there are four matches for the pattern  $\varphi(\rho.a.b \rightarrow \rho.a.c)$  in Figure 4; this XFD can never be satisfied in this tree.

**Lemma 1.**

1. A tree pattern XFD  $A \rightarrow B$  holds on  $T$  if and only if the relational FD  $A \rightarrow B$  holds on  $U(R(T))$ .
2. Tree tuple XFDs [3] are expressible as tree pattern XFDs.

**3.2 The Implication Problem**

Our primary area of focus is the implication problem. Given an XFD  $\sigma$  and document  $T$ , we write  $T \models \sigma$  to mean  $\sigma$  holds in  $T$ . Given a set  $\Sigma$ , we write  $T \models \Sigma$  if every  $\tau \in \Sigma$  holds in  $T$ . Finally, we write  $\Sigma \models \sigma$  to mean that, for all  $T \models \Sigma$ ,  $T \models \sigma$ . Similarly, for a DTD  $D$ ,  $\Sigma \models_D \sigma$  means that, for any  $T \models \Sigma$  and  $T \models D$ , we have  $T \models \sigma$ . Given  $(\Sigma, \sigma)$  [or in the case of a DTD,  $(\Sigma, \sigma, D)$ ], the implication problem is to decide whether  $\Sigma \models \sigma$  [respectively,  $\Sigma \models_D \sigma$ ].

As in the relational literature, we allow implication to be unrestricted (i.e. the documents can be potentially infinite). However, finite and unrestricted implication coincide in the absence of a DTD, and in the presence of a nonrecursive DTD [15]. Dealing with unrestricted implication means that we have to be able to reason about trees of infinite height and/or branching. However, for any particular problem instance, we need only consider a finite subtree of any document. For details, see [15].

In our chase algorithm, we will assume that each instance  $(\Sigma, \sigma)$  is given so that  $\sigma$  has the form  $A \rightarrow b$ , with  $b$  a single path, and all  $C \rightarrow d \in \Sigma$  have a single path on the right-hand side as well. As we will see from the soundness of our axiomatization in Section 4.2, this assumption is without loss of generality. By making it, we can give a clearer presentation and properly compare our complexity results with existing work, which makes the same assumption [3]. If our input is not in this form, conversion is polynomial, so tractability is not affected.

**4 Implication without a DTD**

Before we understand how DTDs affect the XFDs satisfied by a document, we must first understand the theory of XFDs alone.

**4.1 Chase algorithm for XFD implication**

This section presents a fast algorithm for deciding XFD implication. This algorithm is essentially the standard chase algorithm adapted to XML documents. Suppose  $\sigma = A \rightarrow b$ . We set up a tableau  $\mathcal{T}$  as follows: there are two rows, and one column for every path (and prefix of a path) in  $A$  and  $b$ . In the  $\rho$  column and in each column corresponding to a path in  $A$  itself, we insert two identical variables from some indexed set  $\{v_i\}_{i \in I}$ ; however, no variables are repeated between columns. For all other entries in the tableau, we assign unique variables.

Consider, as an example, the dependency  $\{\rho.a.b, \rho.a.c\} \rightarrow \rho.a.d.e$ . This corresponds to the tableau in Figure 5. Next, we define the set  $\widehat{\Sigma}$  of functional dependencies that are used to chase on  $\mathcal{T}$ . We let  $X$  be the set of attributes of  $\mathcal{T}$  (equal to  $\text{paths}(\varphi(\sigma))$ );  $\widehat{\Sigma}$  consists exactly of  $\Sigma$  plus the following additional dependencies:

- $p.d \rightarrow p$  for every  $p.x \in X, x \in \text{EL}$  (non-attribute parents are unique).
- $p \rightarrow p.\alpha$  for every  $p.\alpha \in X$  (attributes have unique values).

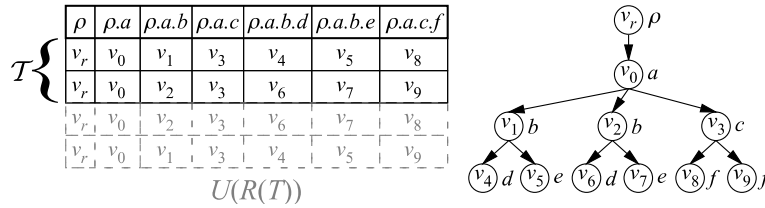
$\rho$	$\rho.a$	$\rho.a.b$	$\rho.a.c$	$\rho.a.d$	$\rho.a.d.e$
$v_1$	$v_2$	$v_3$	$v_4$	$v_5$	$v_7$
$v_1$	$v_9$	$v_3$	$v_4$	$v_6$	$v_8$

**Fig. 5.** Tableau for  $\{\rho.a.b, \rho.a.c\} \rightarrow \rho.a.d.e$

The algorithm now involves chasing with  $\widehat{\Sigma}$ . At each step of the chase we have a dependency  $C \rightarrow d$  and attempt to unify the two variables in the  $d$  column. This is a legal move if either (or both) the following hold:

1. All  $C$  column values of both rows are equal.
2. Let  $q$  be the longest prefix of  $d$  on which the two rows agree. The rows agree on all paths in  $C$  of the form  $q.x.t, x \in \text{EL}$ , where  $q.x$  is a prefix of  $d$ .

The correctness of the first chase rule is clear, as it is the standard one. To understand the second rule, consider the tableau  $\mathcal{T}$  represented by the first two rows in Figure 6, and suppose we are chasing with the dependency  $\{\rho.a.b, \rho.a.c\} \rightarrow \rho.a.c.f$ . This tableau corresponds to the tree on the right hand side of Figure 6. However, when we convert this tree  $T$  back to its relation  $U(R(T))$ , the tree structure gives us two additional rows to the tableau. And from these two new rows we see that we need to unify  $v_8$  and  $v_9$  to satisfy our dependency.



**Fig. 6.** Chasing with  $\{\rho.a.b, \rho.a.c\} \rightarrow \rho.a.c.f$

One way to solve this problem is to construct our tableau so that it is closed under transformation to a tree and back to  $U(R(T))$ . However, the size of this tableau could be exponential in the size of our XFD  $\sigma$ . Fortunately, the regularity of the tree structure makes this unnecessary. We see that the two rows in the tableau for Figure 6 have the same value of  $\rho.a.c$ . As  $\rho.a.b$  is a branch independent of  $\rho.a.c$ , the tree structure ensures that the equality between the values for  $\rho.a.c$  is enough to unify  $\rho.a.c.f$ . The second chase rule generalizes this idea, allowing us to restrict our chase to just two rows.

When the chase terminates (as it clearly must), we can construct a tree  $T_{chase}$  from the tableau. By the following theorem, the values in the two columns for  $d$  are equal if and only if  $\Sigma \models \sigma$ .

**Theorem 1.** *After the chase terminates,  $T_{chase} \models \sigma$  if and only if  $\Sigma \models \sigma$ .*

Computing the chase naively is  $O(|\Sigma||\sigma|)$ . We have at most  $|\sigma|$  iterations of the outer loop, and finding a dependency that satisfies either rule 1 or rule 2 requires a scan of  $\Sigma$ . There is the issue that we are chasing with  $\widehat{\Sigma}$  and not  $\Sigma$ . However, the functional dependencies in  $\widehat{\Sigma} \setminus \Sigma$  depend on a single column and can be attached to that column for constant time evaluation.

It is easy to create a data structure that tracks, for each  $C \rightarrow d \in \Sigma$ , the greatest prefix of  $d$  unified in the chase so far. This allows us to adapt rule 2 to the linear time chase presented in Beeri and Bernstein [5]. We refer the reader to [15] for details.

**Corollary 1.** *Given  $(\Sigma, \sigma)$ , implication can be decided in  $O(|\Sigma| + |\sigma|)$  time.*

## 4.2 Axiomatization

**Nonnull constraints and fictitious functional dependencies** Given the chase algorithm, we can now extract an axiomatization using traditional techniques[1]. Our axiomatization requires that our axiom language be able to express two additional types of constraints on XML documents. The first are the nonnull constraints; intuitively, certain nodes in a tree may not be null provided that certain other nodes are not null. For example, the root may never be null, and every nonnull node must have a nonnull parent. Formally, for  $A, B \subseteq \text{PATHS}$ , we say that  $nn(A, B)$  if in the smallest tree pattern for  $A$  and  $B$ , whenever the paths in  $A$  are not null, neither are the paths in  $B$ .

The second constraint is a variation on the FD concept. Following Atzeni and Morfuni [4], we refer to these constraints as fictitious functional dependencies (FFDs).

**Definition 2.** *A fictitious functional dependency  $\sigma = A \xrightarrow{C} B$  consists of three subsets  $A, B, C \subseteq \text{PATHS}$ . We let  $\varphi(\sigma)$  be the smallest tree pattern in which all paths in  $A, B$ , and  $C$  occur, and let  $=_{fd}$  be as in a normal XFD. The dependency holds if, for any two matches  $\mu_1, \mu_2 \in \mathcal{M}_{\varphi(\sigma), T}$ , whenever we have  $\mu_1(p) =_{fd} \mu_2(p)$  for all  $p \in A$ , and  $\mu_1(s) \neq \text{null}$  for all  $s \in C$ , then for all  $q \in B$ , either  $\mu_1(q) = \mu_2(q) = \text{null}$  or  $\mu_1(q) =_{fd} \mu_2(q)$ .*

It is absolutely essential to note that the condition  $\mu_1(p)$  and  $\mu_2(p) \neq \text{null}$  for all  $p \in A$ , present for ordinary XFDs, is missing from the above definition. Thus the matches are now no longer required to be nonnull on  $A$ , only equal. For example, suppose we add an additional node  $v_{13}$  to the tree in Figure 3; we add this node as a child of  $v_2$  and label it  $e$  (the label of  $v_7$ ). Then this tree satisfies the XFD  $\rho.a.b.f \rightarrow \rho.a.e$  because there is a unique occurrence of each path below  $v_1$ , but it does not satisfy the FFD  $\rho.a.b.f \xrightarrow{\rho.a,b} \rho.a.e$ .

**The Axiomatization** To axiomatize XFDs, we start with the axioms that Atzeni and Morfuni [4] introduced for relation FDs in the presence of null values. We let  $X, Y, Z, W \subseteq \text{PATHS}$ . The following set of 12 axioms are sound and complete for relational FDs in the presence of nulls.

1. If  $Y \subseteq X$ , then  $nn(X, Y)$ .

2. If  $nn(X, Y)$  then  $nn(XZ, YZ)$ .
3. If  $nn(X, Y)$  and  $nn(Y, Z)$  then  $nn(X, Z)$
4. If  $Y \subseteq X$ , then  $X \rightarrow Y$
5. If  $X \rightarrow Y$ , then  $XZ \rightarrow YZ$
6. If  $X \rightarrow Y$  and  $X \rightarrow Z$ , then  $X \rightarrow YZ$
7. If  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$
8. If  $Y \subseteq X \subseteq Z$  then  $X \xrightarrow{Z} Y$  (reflexivity for FFDs)
9. If  $X \xrightarrow{Z} Y$  and  $W \subseteq Z$ , then  $XW \xrightarrow{Z} YW$  (augmentation for FFDs)
10. If  $X \xrightarrow{W} Y$  and  $Y \xrightarrow{W} Z$  then  $X \xrightarrow{W} Z$  (transitivity for FFDs)
11. If  $X \rightarrow Y$  and  $X \subseteq Z$  then  $X \xrightarrow{Z} Y$  (FDs to FFDs)
12. If  $X \xrightarrow{Z} p$  and  $nn(Xp, Z)$  then  $X \rightarrow p$  (FFDs to FDs)

We introduce an additional technical axiom for working with FFDs.

13. If  $X \xrightarrow{Z} Y$  and  $nn(W, Z)$  then  $X \xrightarrow{W} Y$

Furthermore, we need several axioms to capture the tree structure of an XML document. We let  $p, q \in \text{PATHS}$ ,  $q$  is a prefix of  $p$ ,  $x, y \in \text{EL}$ ,  $x \neq \alpha$ .

14.  $nn(\rho)$  (root is never null)
15.  $X \rightarrow \rho$  (root is unique)
16.  $nn(p, q)$  (if a node is not null, neither are any of its ancestors).
17.  $p.x \rightarrow p$  (every non-attribute child has a unique parent).
18.  $p \rightarrow p.\alpha$  (unique attribute child)

Finally, we need one axiom to capture the ability to “splice” paths together as we saw with the second chase rule illustrated in Figure 6. For this axiom, if  $s \in \text{PATHS}$ , let  $s.Y$  denote a set of paths all having  $s$  as a prefix.

19. If  $X, q.Y \xrightarrow{Z} q.y.s$ ,  $q$  not a prefix of any path in  $X$ , where  $X, q.Y \subseteq Z$  and  $q.y.W \subseteq q.Y$  includes all paths in  $q.Y$  with  $q.y$  as a prefix, then  $q, q.y.W \xrightarrow{Z} q.y.s$ .

**Theorem 2.** *Axioms 1-19 are sound and complete for XFD implication without a DTD.*

## 5 XFDs in the presence of a DTD

Our chase algorithm and axiomatization for XFDs can be extended to documents that conform to certain classes of DTDs.

### 5.1 Simple DTDs

**The chase algorithm** It is possible to modify our chase algorithm from Section 4.1 to include the presence of a simple DTD. For a problem instance  $(\Sigma, \sigma)$ , we set up the tableau exactly as before, with a column for every path and prefix of a path in

$\varphi(\sigma)$ . However, in addition, we add a column for every path  $p \in \text{paths}(\Sigma)$  such that  $D$  enforces  $nn(\varphi(\sigma), p)$ .

As before, we define the extension  $\widehat{\Sigma}$  of  $\Sigma$ . However, we also add XFDs of the form  $p.d_1 \rightarrow p.d_1.d_2$  for every column  $p.d_1.d_2$  such that  $D$  contains a production of the form  $d_1 \rightsquigarrow d_2\gamma$  or  $d_1 \rightsquigarrow d_2?\gamma$ ,  $d_2 \notin \gamma$ . These additional XFDs represent the unique child constraints specified by  $D$ .

With this new tableau and  $\widehat{\Sigma}$  we run the chase as before; the values in the  $b$  column are equal if and only if  $\Sigma \models_D \sigma$ . To see this, we construct a tree  $T_{chase}$  from the tableau. Note that this tree may not satisfy the DTD  $D$ ; however, we can extend  $T_{chase}$  to a document  $\langle T_{chase} \rangle_D$  that does.

**Theorem 3.** *After the chase terminates,  $\langle T_{chase} \rangle_D \models \sigma$  if and only if  $\Sigma \models_D \sigma$ .*

The only additional cost for running this chase comes from the additional number of columns and the additional XFDs for the DTD  $D$ . This allows us to improve on the previous quadratic bound obtained in [3] for this variant of the implication problem:

**Corollary 2.** *Given  $(\Sigma, \sigma, D)$  with  $D$  a simple DTD, implication can be decided in  $O(|\Sigma| + |\sigma| + |D|)$  time.*

**Axiomatization** To axiomatize XFD implication for simple DTDs, we need only account for the additional XFDs that we added to  $\widehat{\Sigma}$  in the chase. However, instead of an additional axiom, we add an *axiom schema*. That is, the actual axioms depend on our DTD  $D$ , but we have a single schema for specifying these axioms from the DTD.

**Theorem 4.** *In the presence of a simple DTD  $D$ , a sound and complete set of axioms for (unrestricted) XFD implication includes exactly Axioms 1-19 from Section 4.2 with*

20.  $nn(p.x, p.x.y)$ , for each production  $x \rightsquigarrow y\gamma$ ,  $x \rightsquigarrow y^+\gamma \in D$  (i.e. the DTD does not allow certain children to be null).
21.  $p.x \rightarrow p.x.y$ , for each production  $x \rightsquigarrow y\gamma$ ,  $x \rightsquigarrow y?\gamma \in D$ , where  $\gamma \in (\text{EL} - \{y\})^*$  (i.e. XFDs enforced by the DTD).

## 5.2 #-DTDs

The implication problem in the presence of #-DTDs is more complicated because it is possible for these DTDs to be inconsistent with an XFD. Consider the DTD of the example in Figure 1, with the XFD  $p \rightarrow q$ , where  $p$  is a path identifier for a patient, and  $q$  is a path identifier for a recommending doctor. This XFD is inconsistent with our DTD; there is no document that can satisfy both the DTD and the functional dependency.

For many classes of DTDs, consistency is trivial, in the sense that there is no set of XFDs inconsistent with the DTD. #-DTDs are the lowest class in our hierarchy for which consistency becomes an issue. Consistency and implication are in fact related problems, as the former can be reduced to the latter [11].

However, in our extension of the chase algorithm and our axiomatization to #-DTDs, we solve the consistency problem directly. In particular, given an instance  $(\Sigma, \sigma, D)$  of an implication problem, if  $\Sigma$  and  $D$  are inconsistent then  $\Sigma \models_D \sigma$  is trivially true.

Therefore, our algorithm for deciding implication in the presence of #-DTDs starts by checking for inconsistency directly, in order to rule out this case.

To make this possible, we introduce a quadratic time algorithm for checking consistency of a set of XFDs  $\Sigma$  with a #-DTD. In this paper, we only present a naïve exponential algorithm. The details of the polynomial algorithm can be found in [15]. Our algorithm rests on the fact that issues of consistency can be decided on the smallest tree satisfying  $D$ .

**Lemma 2.** *Let  $T_D$  be the smallest tree such that  $T_D \models D$ .  $(\Sigma, D)$  is consistent if and only if we can assign attribute values to any  $\alpha$ -nodes in  $T_D$  so that  $T_D \models \Sigma$ .*

With this in mind, our algorithm is as follows:

1. Construct the minimal  $T_D$  from  $D$ . Initialize all the attribute nodes to different values from VAL, and convert  $T_D$  to the flat relation  $U(R(T_D))$ .
2. Partition  $\Sigma$  into  $\Sigma_a$ , which contains all the XFDs with an attribute node on the right-hand side, and  $\Sigma_i = \Sigma \setminus \Sigma_a$ .
3. Treat  $U(R(T_D))$  as a very large chase tableau, and perform a standard relational chase (no use of the second rule) on it with  $\Sigma$ . Any unification from  $\Sigma_a$  is allowed to proceed normally, but any attempt at a unification from  $\Sigma_i$  causes the algorithm to return “no”.
4. If the chase completes successfully, return “yes”.

**Lemma 3.** *There is an assignment of attribute values to attribute nodes of  $T_D$  under which  $T_D \models \Sigma$  if and only if the algorithm returns “yes”.*

This algorithm is possibly exponential because of the duplicate paths that may appear in  $T_D$ , resulting in a large  $U(R(T_D))$ . However, using special encoding techniques to compress duplicate paths, we can implement this algorithm in polynomial time.

**Proposition 1.** *Given a pair  $(\Sigma, D)$  where  $D$  is a #-DTD, checking consistency requires  $O(|D| + |\Sigma|^2)$  time.*

**The chase algorithm** As we mentioned above, our first step in handling any implication problem instance  $(\Sigma, \sigma, D)$ , is to determine whether  $(\Sigma, D)$  is consistent. A second special situation that can arise is that, while  $\Sigma$  is consistent with  $D$ , we cannot have a tree  $T$  such that  $T \models D$ ,  $T \models \Sigma$  and  $T$  contains even one nonnull match for all of  $\varphi(\sigma)$ . This can happen, for instance, in the case where  $D$  has productions  $\rho \rightsquigarrow a^*b$ ,  $a \rightsquigarrow cc$ ,  $\Sigma = \{\rho.a \rightarrow \rho.a.c\}$  and  $\sigma = \rho.b \rightarrow \rho.a$ . Clearly there is only one tree satisfying  $D$  and  $\Sigma$ : the tree consisting of the root and one  $b$  child. On this one tree,  $\sigma$  also holds, precisely because there is no nonnull match for all of  $\varphi(\sigma)$ . In cases like this one also,  $\Sigma \models_D \sigma$  vacuously. Fortunately, we can adapt our consistency checking algorithm to check for this case as well, without affecting the complexity.

Having ruled out the above two “pathological” cases, we can decide implication by using exactly the same chase that we used for simple DTDs. That gives us the result below. This proof of this theorem is quite technical; see [15].

**Theorem 5.** *Given  $(\Sigma, \sigma, D)$  where  $D$  is a #-DTD, implication can be decided in  $O(|\Sigma|^2 + |\sigma| + |D|)$  time.*

**Axiomatization** To axiomatize implication for #-DTDs, we need to account for the two pathological cases described above. Mathematically, the pathological cases arise because the DTD prevents certain XFDs from holding unless all matches of the left-hand side are null. Again, this requires the introduction of an axiom schema.

**Theorem 6.** *In the presence of a simple DTD  $D$ , a sound and complete set of axioms for (unrestricted) XFD implication includes exactly Axioms 1-21 from Section 5.1 with*

22. *If  $p.x \xrightarrow{X} p.x.y$ , then for  $W \xrightarrow{X.p.x} Z$ , for all  $W, Z$  and all  $x \rightsquigarrow yy\gamma \in D$ .*

### 5.3 Arbitrary DTDs

It is known that the presence of even very limited disjunction in a DTD makes the complexity of the implication problem rise to co-NP complete. For arbitrary DTDs, implication is co-NP hard and computable in co-NEXPTIME, while consistency is NP-hard and computable in NEXPTIME [2].

As for axiomatization, Arenas and Libkin [3] have shown that axiomatization is also difficult in the presence of disjunction; they prove that in their axiom language (which is less rich than ours) no finite axiomatization is possible for arbitrary DTDs. We note, however, that this result need not carry over to all possible axiom languages.

## 6 Related work

In the relational model, functional dependencies [1] and chase algorithms [1, 5] have been studied extensively. Our axiomatization draws on the theory developed for relational FDs in the presence of null values [4, 17]. Functional dependencies have also been studied for nested relations [12].

In XML, functional dependencies have attracted much research attention [3, 20, 14, 19, 16, 13]; see [21] for a survey. Our work uses a definition of XFD that is equivalent to the one in [3]; we have highlighted throughout the paper how our results extend these. The other FD definitions in the literature differ from ours in several ways. However, we believe that our definitions allow us to strike a good balance between generality of the framework and the ability to obtain strong tractability and axiomatization results.

Finally, there has been much work on other constraints in XML documents, notably keys [7], foreign keys [11], path constraints [8] and XICs [9]. For a recent, general survey on XML constraints, see [10]. DTDs are known to interact in a complex fashion with keys [11] and to affect the satisfiability and containment of XPath queries [18, 6].

## 7 Future directions

Several areas emerge as obvious directions for future work. While we know that axiomatization and the implication problem become intractable in the presence of disjunction, these problems are still open for DTDs with nested operations. In addition, the language used in the definition of XFDs can be made richer, for instance by adding wildcard descendant navigation to our tree patterns. Furthermore, our investigation of XFD interaction with DTDs can be broadened to include other popular XML constraint specifications, such as XML Schema.

## 7.1 Acknowledgments

We thank Al Demers, Johannes Gehrke, Dexter Kozen, David Martin, Millist Vincent and the anonymous reviewers for valuable suggestions that helped improve this paper.

## References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. M. Arenas. *Design Principles for XML Data*. PhD thesis, University of Toronto, 2005.
3. M. Arenas and L. Libkin. A normal form for XML documents. *ACM TODS*, 29(1):195–232, 2004.
4. P. Atzeni and N. Morfuni. Functional dependencies and constraints on null values in database relations. *Information and Control*, 70(1):1–31, 1986.
5. C. Beeri and P. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM TODS*, 4(1):pp. 30 – 59, 1979.
6. Michael Benedikt, Wenfei Fan, and Floris Geerts. XPath satisfiability in the presence of DTDs. In *Proc. PODS*, pages 25–36, 2005.
7. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Reasoning about keys for XML. *Inf. Syst.*, 28(8):1037–1063, 2003.
8. P. Buneman, W. Fan, and S. Weinstein. Path constraints in semistructured and structured databases. In *Proc. PODS*, pages 129–138, 1998.
9. A. Deutsch and V. Tannen. Containment and integrity constraints for XPath. In *KRDB*, 2001.
10. W. Fan. XML constraints: Specification, analysis, and applications. In *Proc. DEXA*, pages 805–809, 2005.
11. W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. In *Proc. PODS*, 2001.
12. C. Hara and S. Davidson. Reasoning about nested functional dependencies. In *Proc. PODS*, pages 91–100, 1999.
13. S. Hartmann and S. Link. More functional dependencies for XML. In *ADBS*, pages 355–369, 2003.
14. S. Hartmann and S. Link. On functional dependencies in advanced data models. In *Electronic Notes in Theoretical Computer Science*, volume 84. Elsevier Science B. V., 2003.
15. Ł. Kot and W. White. Characterization of XML functional dependencies and their interaction with DTDs. Technical Report 2006-2039, Cornell University, July 2006.
16. M. Lee, T. Ling, and W. Low. Designing functional dependencies for XML. In *Proc. EDBT*, pages 124–141, 2002.
17. M. Levene and G. Loizou. Axiomatisation of functional dependencies in incomplete relations. *Theor. Comput. Sci.*, 206(1-2):283–300, 1998.
18. F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *Proc. ICDT*, pages 315–329, 2003.
19. K. Schewe. Redundancy, dependencies and normal forms for XML databases. In *ADC*, pages 7–16, 2005.
20. M. Vincent, J. Liu, and C. Liu. Strong functional dependencies and their application to normal forms in XML. *ACM TODS*, 29(3):445–462, 2004.
21. J. Wang. A comparative study of functional dependencies for XML. In *APWeb*, pages 308–319, 2005.