

# Unlocking Keys for XML Trees

Sven Hartmann and Sebastian Link\*

Information Science Research Centre, Massey University, New Zealand  
{s.hartmann,s.link}@massey.ac.nz

**Abstract.** We review key constraints in the context of XML as introduced by Buneman et al. We show that one of the proposed inference rules is not sound in general, and the axiomatisation proposed for XML keys is incomplete even if key paths are simple. Therefore, the axiomatisation and also the implication problem for XML keys are still unsolved.

We propose a set of inference rules that is indeed sound and complete for the implication of XML keys with simple key paths. Our completeness proof enables us to characterise the implication of XML keys in terms of the reachability problem of nodes in a digraph. This results in a quadratic time algorithm for deciding XML key implication, and shows that reasoning for XML keys is practically efficient.

## 1 Introduction

The eXtensible markup language (XML,[6]) has recently evolved to the standard for data exchange on the Web, and also represents a uniform model for data integration. It provides a high degree of syntactic flexibility but has little to offer to specify the semantics of its data. Consequently, the study of integrity constraints has been recognised as one of the most important yet challenging areas of XML research [15,28,31,33]. The importance of XML constraints is due to a wide range of applications ranging from schema design, query optimisation, efficient storing and updating, data exchange and integration, to data cleaning [15]. Therefore, several classes of integrity constraints have been defined for XML including keys [8], path constraints [10,11], inclusion constraints [16,17] and functional dependencies [3,20,22,32]. However, for almost all classes of constraints the complex structure of XML data results in decision problems that are intractable. It is therefore a major challenge to find natural and useful classes of XML constraints that can be reasoned about efficiently [15,16,17,28,31]. Prime candidates of such classes are absolute and relative keys [8,9] that are defined independently from any specification such as a DTD [6] or XSD [30]. Keys are based on the representation of XML data as trees. This is commonly used by DOM [2], XSL [24], and XML Schema [30]. Figure 1 shows such a representation in which nodes are annotated by their type: *E* for element, *A* for attribute, and *S* for string (PCDATA).

---

\* This research is supported by Marsden Funding, the Royal Society of New Zealand.

Keys are defined in terms of path expressions, and determine nodes either relative to a set of context nodes or the root. Nodes are determined by (complex) values on some selected subnodes. In Figure 1, an example of a reasonable absolute key is that the *isbn* values identify the *book* node. That is, the *isbn* subnodes of different *book* nodes must have different values.

In contrast, an *author* cannot be identified in the entire tree by its *first* and *last* subnodes since the same author can have written more than one book. However, the *author* can indeed be identified by its *first* and *last* subnodes relative to the *book* node. That is, for each individual *book* node, different *author* subnodes must differ on their *first* or *last* subnodes.

```

<db>
  <book isbn=0198532741>
    <title>Toposes and Local Set Theories</title>
    <author><first>John</first><last>Bell</last></author>
  </book>
  <book isbn=0720428440>
    <title>A course in mathematical logic</title>
    <author><first>John</first><last>Bell</last></author>
    <author><first>Moshe</first><last>Machover</last></author>
  </book>
</db>

```

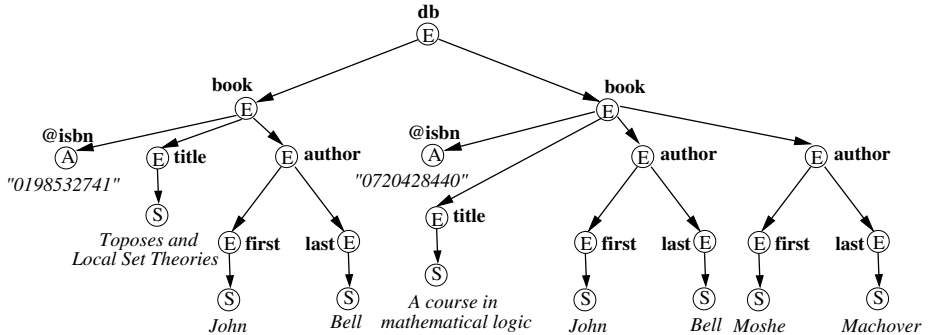


Fig. 1. XML data fragment and its tree representation

**Contributions.** We review XML key constraints. We show that one of the inference rules for key implication [9] is only sound for keys with simple key path expressions [8], but not sound in general as stated in [9]. The incorrectness is not just a minor detail but shows that the choice of a path language for defining XML keys can be crucial. We demonstrate that the axiomatisation proposed in [9] is not only incomplete in the general case but already incomplete in the case of simple key path expressions, i.e., for XML keys as defined in [8].

Since keys have had a significant impact on XML [8,9] we believe that it is important to provide an axiomatisation and show that automated reasoning about XML keys is practically efficient. We propose an axiomatisation of XML keys with simple key path expressions. Our completeness proof is based on a

characterisation of key implication in terms of reachability of nodes in a digraph. Utilising the efficient evaluation of Core XPath queries [18] this results in a decision procedure for XML key implication that is quadratic in the size of the keys given. The simplicity of our algorithm is a result of the technique used for proving the completeness of our inference rules. Notice that the original (flawed) technique resulted in a heptic ( $n^7$ ) algorithm, cf. [9].

**Related Work.** Constraints have been extensively studied in the context of the relational model of data, some excellent surveys include [14,29]. Dependencies have also been investigated in nested data models, cf. [19,21,27]. Recent work on XML constraints include [3,4,8,9,10,11,16,17,32], for a brief survey see [15].

## 2 Prerequisites

We review the definition of keys and their properties [8,9]. Throughout the paper we assume familiarity with basic concepts from graph theory [23].

### 2.1 The XML Tree Model

XML documents can be modelled as node-labelled trees. We assume that there is a countably infinite set  $\mathbf{E}$  denoting element tags, a countably infinite set  $\mathbf{A}$  denoting attribute names, and a singleton  $\{S\}$  denoting text (PCDATA). We further assume that these sets are pairwise disjoint, and put  $\mathcal{L} = \mathbf{E} \cup \mathbf{A} \cup \{S\}$ . We refer to the elements of  $\mathcal{L}$  as *labels*.

An *XML tree* is a 6-tuple  $T = (V, lab, ele, att, val, r)$  where  $V$  denotes a set of nodes;  $lab$  is a mapping  $V \rightarrow \mathcal{L}$  assigning a label to every node in  $V$ ; a node  $v$  in  $V$  is called an *element (E) node* if  $lab(v) \in \mathbf{E}$ , an *attribute (A) node* if  $lab(v) \in \mathbf{A}$ , and a text (S) node if  $lab(v) = S$ ;  $ele$  and  $att$  are partial mappings defining the edge relation of  $T$ : for any node  $v$  in  $V$ , if  $v$  is an element node, then  $ele(v)$  is a list of element and text nodes in  $V$  and  $att(v)$  is a set of attribute nodes in  $V$ ; if  $v$  is an attribute or text node then  $ele(v)$  and  $att(v)$  are undefined;  $val$  is a partial mapping assigning a string to each attribute and text node: for any node  $v$  in  $V$ , if  $v$  is an  $A$  or  $S$  node then  $val(v)$  is a string, and  $val(v)$  is undefined otherwise; and  $r$  is the unique and distinguished root node.

An XML tree is said to be *finite* if  $V$  is finite. For a node  $v \in V$ , each node  $w$  in  $ele(v)$  or  $att(v)$  is called a *child* of  $v$ , and we say that there is an edge  $(v, w)$  from  $v$  to  $w$  in  $T$ . An XML tree has a tree structure: for each node  $v \in V$ , there is a unique (directed) path of edges from the root  $r$  to  $v$ .

We can now define value equality for pairs of nodes in XML trees. Informally, two nodes  $u$  and  $v$  of an XML tree  $T$  are value equal if they have the same label and, in addition, either they have the same string value if they are  $S$  or  $A$  nodes, or their children are pairwise value equal if they are  $E$  nodes. More formally, two nodes  $u, v \in V$  are *value equal*, denoted by  $u =_v v$ , if and only if the following conditions are satisfied:

- (a)  $lab(u) = lab(v)$ ,
- (b) if  $u, v$  are  $A$  or  $S$  nodes, then  $val(u) = val(v)$ ,
- (c) if  $u, v$  are  $E$  nodes, then (i) if  $att(u) = \{a_1, \dots, a_m\}$ , then  $att(v) = \{a'_1, \dots, a'_m\}$  and there is a permutation  $\pi$  on  $\{1, \dots, m\}$  such that  $a_i =_v a'_{\pi(i)}$  for  $i = 1, \dots, m$ , and (ii) if  $ele(u) = [u_1, \dots, u_k]$ , then  $ele(v) = [v_1, \dots, v_k]$  and  $u_i =_v v_i$  for  $i = 1, \dots, k$ .

That is, two nodes  $u$  and  $v$  are value equal whenever the subtrees rooted at  $u$  and  $v$  are isomorphic by an isomorphism that is the identity on string values. For example, the first and second *author* node (according to document order) in Figure 1 are value equal.

## 2.2 Path Languages

In order to define keys we need a path language that is expressive enough to be practical, yet sufficiently simple to be reasoned about efficiently. This is the case for the path languages  $PL_s$  and  $PL$  [8,9].

**Table 1.** The path languages  $PL_s$  and  $PL$

Path Language	Syntax
$PL_s$	$P ::= \varepsilon \mid \ell.P$
$PL$	$Q ::= \varepsilon \mid \ell \mid Q.Q \mid \_*$

A *path expression* is a (possibly empty) finite list of symbols. In this paper, a *simple path expression* is a path expression that consists of labels from  $\mathcal{L}$ . We use the languages  $PL_s$  and  $PL$  to describe path expressions. Both languages are fragments of regular expressions.  $PL_s$  expressions and  $PL$  expressions are defined by the grammars in Table 1. Herein,  $\varepsilon$  denotes the empty path expression, “.” denotes the concatenation of two path expressions, and  $\ell$  denotes any element of  $\mathcal{L}$ . The language  $PL$  is a generalisation of  $PL_s$  that allows the distinguished symbol “ $\_*$ ” to occur. We call  $\_*$  the *don't care* symbol. It serves as a combination of the wildcard “ $\_$ ” and the Kleene star “ $*$ ”. Note that every  $PL_s$  expression is a  $PL$  expression, too.

We now introduce some more terminology [8,9] used throughout this paper. Note that for  $PL$  expressions, the equalities  $Q.\varepsilon = \varepsilon.Q = Q$  for all  $Q \in PL$ , and  $\_*. \_*$  =  $\_*$  hold. A  $PL$  expression  $Q$  in *normal form* [8,9] does not contain consecutive  $\_*$ , and it does not contain  $\varepsilon$  unless  $Q = \varepsilon$ . A  $PL$  expression can be transformed into normal form in linear time, just by removing superfluous  $\_*$  and  $\varepsilon$  symbols. To simplify discussion, we usually assume that  $PL$  expressions are already given in normal form. The *length*  $|Q|$  of a  $PL$  expression  $Q$  is the number of labels in  $Q$  plus the number of  $\_*$  in the normal form of  $Q$ . The empty path expression  $\varepsilon$  has length 0.

When replacing all  $\_*$  in a  $PL$  expression  $Q$  by simple path expressions, we obtain a simple path expression  $P$  and write  $P \in Q$ . Thus, a  $PL$  expression  $Q$  gives rise to a regular language of simple path expressions  $P \in Q$ . In particular,

a  $PL_s$  expression represents a single simple path expression. For convenience, we will sometimes refer to  $PL_s$  expressions as simple path expressions, too.

We will use path expressions to describe sets of paths in an XML tree  $T$ . Recall that each attribute or text node is a leaf in  $T$ . Therefore, a path expression is said to be *valid* if it does not contain a label  $\ell \in \mathbf{A}$  or  $\ell = S$  in a position other than the last one. In the sequel, we use a valid  $PL$  expression to represent only valid simple path expressions. For example,  $_* .author$  is a valid  $PL$  expression that represents (among others) the valid simple path expression  $book.author$ .

A path  $p$  is a sequence of pairwise distinct nodes  $v_0, \dots, v_m$  where  $(v_{i-1}, v_i)$  is an edge for  $i = 1, \dots, m$ . We call  $p$  a path from  $v_0$  to  $v_m$ , and say that  $v_m$  is *reachable* from  $v_0$  following the path  $p$ . The path  $p$  gives rise to a valid simple path expression  $lab(v_1) \dots lab(v_m)$ , which we denote by  $lab(p)$ . Let  $P$  be a simple path expression, and  $Q$  a  $PL$  expression. A path  $p$  is called a  $P$ -path if  $lab(p) = P$ , and a  $Q$ -path if  $lab(p) \in Q$ . If  $p$  is a path from  $v$  to  $w$ , then  $w$  is said to be reachable from  $v$  following a  $P$ -path or  $Q$ -path, respectively. We also write  $T \models P(v, w)$  and  $T \models Q(v, w)$  when  $w$  is reachable from  $v$  following a  $P$ -path or  $Q$ -path, respectively, in an XML tree  $T$ . For example, in the XML tree in Figure 1, all *first* nodes are reachable from the root following a  $book.author.first$ -path. Consequently, they are also reachable from the root following a  $_* .first$ -path.

For a node  $v$  of an XML tree  $T$ , let  $v[[Q]]$  denote the set of nodes in  $T$  that are reachable from  $v$  by following the  $PL$  expression  $Q$ , i.e.,  $v[[Q]] = \{w \mid T \models Q(v, w)\}$ . We shall use  $[[Q]]$  as an abbreviation for  $r[[Q]]$  where  $r$  is the root of  $T$ . For example, let  $v$  be the second *book* node in Figure 1. Then  $v[[author]]$  is the set of all *author* nodes that are children of the second book. Furthermore,  $[[_* .author]]$  is the set of all *author* nodes in the entire XML tree.

For nodes  $v$  and  $v'$  of  $T$ , the *value intersection* of  $v[[Q]]$  and  $v'[[Q]]$  is given by  $v[[Q]] \cap_v v'[[Q]] = \{(w, w') \mid w \in v[[Q]], w' \in v'[[Q]], w =_v w'\}$  [9]. That is,  $v[[Q]] \cap_v v'[[Q]]$  consists of all those node pairs in  $T$  that are value equal and are reachable from  $v$  and  $v'$ , respectively, by following  $Q$ -paths.

A  $PL$  expression  $Q$  is said to be *contained* in a  $PL$  expression  $Q'$ , denoted by  $Q \subseteq Q'$ , if for any XML tree  $T$  and any node  $v$  of  $T$  we have  $v[[Q]] \subseteq v[[Q']]$ . That is, every node that is reachable from  $v$  by following a  $Q$ -path is also reachable from  $v$  by following a  $Q'$ -path. Note that  $Q \subseteq Q'$  holds if and only if every valid path expression  $P \in Q$  satisfies  $P \in Q'$  [9]. The *containment problem* of  $PL$  is to decide, given any  $PL$  expressions  $Q$  and  $Q'$ , whether  $Q \subseteq Q'$  holds. The containment problem of  $PL$  is decidable in  $\mathcal{O}(|Q| \times |Q'|)$  time [9].

Note that although we have presented the language  $PL$  using the syntax of regular expressions, there is an easy conversion of  $PL$  expressions to  $XPath$  [12] expressions, just by replacing “ $_*$ ” of a  $PL$  expression with “ $/$ ”, and “ $.$ ” with “ $/$ ”. Also, if a  $PL$  expression is meant to start from the root, the converted path is preceded with the symbol “ $/$ ”.

The choice of a path language is directly influenced by the complexity of its containment problem. Buneman et al. [8,9] argue that  $PL$  is simple yet expressive enough to be adopted by XML designers and maintained by systems for XML applications.

### 2.3 Keys for XML

In [9], Buneman et al. define a key  $\varphi$  as an expression  $(Q, (Q', \{Q_1, \dots, Q_k\}))$  where  $Q, Q', Q_i$  are *PL* expressions such that  $Q.Q'.Q_i$  is a valid *PL* expression for all  $i = 1, \dots, k$ . Herein,  $Q$  is called the *context path*,  $Q'$  is called the *target path*, and  $Q_1, \dots, Q_k$  are called the *key paths* of  $\varphi$ .

An XML tree  $T$  satisfies the key  $(Q, (Q', \{Q_1, \dots, Q_k\}))$  if and only if for any node  $q \in \llbracket Q \rrbracket$  and any nodes  $q'_1, q'_2 \in q \llbracket Q' \rrbracket$  such that there are nodes  $x_i \in q'_1 \llbracket Q_i \rrbracket, y_i \in q'_2 \llbracket Q_i \rrbracket$  with  $x_i =_v y_i$  for all  $i = 1, \dots, k$ , then  $q'_1 = q'_2$  [9]. More formally,  $\forall q \in \llbracket Q \rrbracket \forall q'_1, q'_2 \in q \llbracket Q' \rrbracket$

$$\left( \bigwedge_{1 \leq i \leq k} q'_1 \llbracket Q_i \rrbracket \cap_v q'_2 \llbracket Q_i \rrbracket \neq \emptyset \right) \Rightarrow q'_1 = q'_2.$$

Moreover, Buneman et al. [9] present a finite set of inference rules which they state is sound and complete for the implication of XML keys, as well as an implication decision algorithm based on this axiomatisation.

**Table 2.** An axiomatisation of XML keys whose key paths are *PL<sub>s</sub>* expressions

$\frac{}{(Q, (\varepsilon, S))}$ (epsilon)	$\frac{(Q, (Q', S \cup \{\varepsilon, P\}))}{(Q, (Q', S \cup \{\varepsilon, P.P'\}))}$ (prefix-epsilon)	$\frac{(Q, (Q', S))}{(Q, (Q', S \cup \{P\}))}$ (superkey)
$\frac{(Q, (Q'.P, \{P'\}))}{(Q, (Q', \{P.P'\}))}$ (subnodes)	$\frac{(Q, (Q', S))}{(Q'', (Q', S))} Q'' \subseteq Q$ (context-path-containment)	$\frac{(Q, (Q', S))}{(Q, (Q'', S))} Q'' \subseteq Q'$ (target-path-containment)
$\frac{(Q, (Q'.Q'', S))}{(Q.Q', (Q'', S))}$ (context target)	$\frac{(Q, (Q'.P, \{\varepsilon, P'\}))}{(Q, (Q', \{\varepsilon, P.P'\}))}$ (subnodes-epsilon)	$\frac{(Q, (Q', \{P.P_1, \dots, P.P_k\})), (Q.Q', (P, \{P_1, \dots, P_k\}))}{(Q, (Q'.P, \{P_1, \dots, P_k\}))}$ (interaction)

Let  $\mathfrak{S}$  denote the set of inference rules from Table 2 without the *subnodes-epsilon rule*. This is the axiomatisation proposed in [9] when only *PL<sub>s</sub>* expressions are allowed for the key paths. Unfortunately,  $\mathfrak{S}$  turns out to be incomplete since the *subnodes-epsilon rule* is sound for the implication of XML keys and independent from  $\mathfrak{S}$ .

The soundness of the *subnodes-epsilon rule* is not difficult to see. Suppose an XML tree  $T$  violates  $(Q, (Q', \{\varepsilon, P.P'\}))$ . Then there is some node  $q \in \llbracket Q \rrbracket$  and some nodes  $q'_1, q'_2 \in q \llbracket Q' \rrbracket$  such that  $q'_1 \neq q'_2$ ,  $q'_1 =_v q'_2$ , and there exist  $p'_1 \in q'_1 \llbracket P.P' \rrbracket$  and  $p'_2 \in q'_2 \llbracket P.P' \rrbracket$  such that  $p'_1 =_v p'_2$ . By definition, there exists some  $p_1 \in q'_1 \llbracket P \rrbracket$  such that  $p'_1 \in p_1 \llbracket P' \rrbracket$ . Since  $q'_1 =_v q'_2$  it is easy to see that there exists some node  $p_2 \in q'_2 \llbracket P \rrbracket$  such that  $p_1 \neq p_2$ ,  $p_1 =_v p_2$  and  $p'_2 \in p_2 \llbracket P' \rrbracket$ . But then  $p_1 \in q \llbracket Q'.P \rrbracket$  and  $p_2 \in q \llbracket Q'.P \rrbracket$ . Hence,  $T$  also violates  $(Q, (Q'.P, \{\varepsilon, P'\}))$ .

Moreover, the *subnodes-epsilon* rule is independent from  $\mathfrak{S}$ , i.e., there is a finite set  $\Sigma \cup \{\varphi\}$  of XML keys such that  $\varphi$  cannot be derived from  $\Sigma$  by  $\mathfrak{S}$ , but  $\varphi$  can be derived from  $\Sigma$  using  $\mathfrak{S}$  and the *subnodes-epsilon* rule. A simple example is given by  $\Sigma = \{(\varepsilon, (A.B, \{\varepsilon, C\}))\}$  and  $\varphi = (\varepsilon, (A, \{\varepsilon, B.C\}))$ . In fact, *subnodes* is the only inference rule in  $\mathfrak{S}$  that allows us to make an infix of the target path in the premise of a rule a prefix of a key path in the conclusion of the rule. Since *subnodes* only permits a singleton as the set of key paths, and the other inference rules do not allow us to generate conclusions with a single key path from premises that have at least two key paths, it is impossible to derive  $\varphi$  from  $\Sigma$  using  $\mathfrak{S}$ . On the other hand, however,  $\varphi$  can be inferred from  $\Sigma$  by a single application of *subnodes-epsilon*. The soundness of the *subnodes-epsilon* rule shows that  $\varphi$  is implied by  $\Sigma$ , but  $\varphi$  cannot be inferred from  $\Sigma$  by  $\mathfrak{S}$ . Consequently,  $\mathfrak{S}$  is incomplete, even for the implication of XML keys with  $PL_s$  expressions as key paths.

In the more general case where one allows key paths to be in  $PL$  the *subnodes* rule

$$\frac{(Q, (Q'.Q'', \{P\}))}{(Q, (Q', \{Q''.P\}))}$$

is not even sound for the implication of XML keys. A simple counter-example is the XML tree  $T$  illustrated in Figure 2.  $T$  satisfies the absolute key  $\sigma = (\varepsilon, (a.-*.b.c.-*.d, \{e\}))$ , but violates the absolute key  $\varphi = (\varepsilon, (a.-*.b, \{c.-*.d.e\}))$  since  $v_3, v_6 \in \llbracket a.-*.b \rrbracket$ ,  $v_3 \neq v_6$  and  $v_3 \llbracket c.-*.d.e \rrbracket \cap_v v_6 \llbracket c.-*.d.e \rrbracket = \{(v_{10}, v_{10})\}$ , i.e.,  $\varphi$  is not implied by  $\sigma$ . However,  $\varphi$  can be inferred from  $\sigma$  using the *subnodes* rule. Therefore, the inference rules proposed in [9] are not sound for the implication of keys as defined in [9]. Unfortunately, this is not just a minor detail since the completeness proof [9] makes use of the *subnodes* rule and is therefore not correct. Consequently, there is no completeness proof at all. The conference paper [7] contains a much more restrictive definition of value intersection which leaves the *subnodes* rule sound in the presence of arbitrary  $PL$  expressions for the key paths, but this does not affect the incorrectness of the results in the journal paper [9].

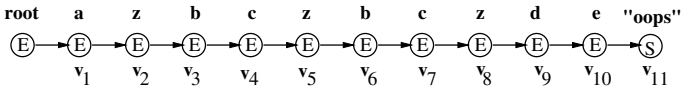


Fig. 2. The *subnodes* rule is not sound for key paths in  $PL$

In this paper we study the implication of XML keys with simple key paths.

**Definition 1.** A key constraint  $\varphi$  for XML (or short XML key) is an expression  $(Q, (Q', S))$  where  $Q, Q'$  are  $PL$  expressions and  $S$  is a non-empty finite set of  $PL_s$  expressions such that  $Q.Q'.P$  are valid  $PL$  expressions for all  $P$  in  $S$ . Herein,  $Q$  is called the context path,  $Q'$  is called the target path, and the elements of  $S$  are called the key paths of  $\varphi$ . If  $Q = \varepsilon$ , we call  $\varphi$  an absolute key; otherwise  $\varphi$  is called a relative key.  $\square$

Let  $\mathcal{K}$  denote the language of XML keys. For an XML key  $\varphi$ , we use  $Q_\varphi$  to denote its context path,  $Q'_\varphi$  to denote its target path, and  $P_1^\varphi, \dots, P_{k_\varphi}^\varphi$  to denote its key paths, where  $k_\varphi$  is the number of its key paths. The size  $|\varphi|$  of a key  $\varphi$  is defined as the sum of the lengths of all path expressions in  $\varphi$ , i.e.,  $|\varphi| = |Q_\varphi| + |Q'_\varphi| + \sum_{i=1}^{k_\varphi} |P_i^\varphi|$ .

*Example 1.* We formalise the examples from the introduction. In an XML tree that satisfies the absolute key  $(\varepsilon, (-*.book, \{isbn\}))$  one will never be able to find two different *book* nodes that have value equal *isbn* subnodes. Furthermore, under a *book* node in an XML tree that satisfies the relative key  $(-*.book, (author, \{first, last\}))$  one will never find two different *author* subnodes that are value equal on their *first* and *last* subnodes.  $\square$

It is stated in [8] “that allowing arbitrary path expressions for the  $P_i$  [key paths] merely complicates the definition of key but does not change much in the way of the theory”. This is not true since the *subnodes* rule is sound according to the original XML key definition [8] but not if the  $P_i$ s are arbitrary *PL* expressions as the example above shows.

We will prove that the inference rules from [9] together with the *subnodes-epsilon* rule are sound and complete for the implication of XML keys as defined in Definition 1.

**Theorem 1.** *The inference rules from Table 2 are sound and complete for the implication of XML keys in  $\mathcal{K}$ .*  $\square$

### 3 An Axiomatisation

Let  $\Sigma$  be a finite set of keys in  $\mathcal{K}$ . An XML tree  $T$  satisfies  $\Sigma$  if and only if  $T$  satisfies every  $\sigma \in \Sigma$ . Let  $\Sigma \cup \{\varphi\}$  be a finite set of keys in  $\mathcal{K}$ . We say that  $\Sigma$  (*finitely*) *implies*  $\varphi$ , denoted by  $\Sigma \models_{(f)} \varphi$ , if and only if every (finite) XML tree  $T$  that satisfies  $\Sigma$  also satisfies  $\varphi$ . The (*finite*) *implication problem* is to decide, given any finite set of keys  $\Sigma \cup \{\varphi\}$ , whether  $\Sigma \models_{(f)} \varphi$ . For a set  $\Sigma$  of keys in  $\mathcal{K}$ , let  $\Sigma^* = \{\varphi \in \mathcal{K} \mid \Sigma \models \varphi\}$  be its *semantic closure*, i.e., the set of all keys implied by  $\Sigma$ . Finite and unrestricted implication problem coincide for the class of keys in  $\mathcal{K}$  [9]. We will therefore commonly speak of the *implication problem* for keys in  $\mathcal{K}$ .

The notion of derivability ( $\vdash_{\mathfrak{R}}$ ) with respect to a set  $\mathfrak{R}$  of inference rules can be defined analogously to the notion in the relational data model [1, pp. 164-168]. For a set  $\Sigma$  of keys in  $\mathcal{K}$ , let  $\Sigma_{\mathfrak{R}}^+ = \{\varphi \mid \Sigma \vdash_{\mathfrak{R}} \varphi\}$  be its *syntactic closure* under inference using  $\mathfrak{R}$ .

The aim in this section is to demonstrate that the set  $\mathfrak{R}$  of inference rules in Table 2 is *sound* (i.e.,  $\Sigma_{\mathfrak{R}}^+ \subseteq \Sigma^*$ ) and *complete* (i.e.,  $\Sigma^* \subseteq \Sigma_{\mathfrak{R}}^+$ ) for the implication of keys as defined in Definition 1.

Our completeness argument is fundamentally different from the technique proposed in [9]. Buneman et al. [9] apply the following strategy: for any  $\varphi \notin \Sigma^+$  they construct a finite XML tree  $T$  that violates  $\varphi$ . Subsequently, they *chase*

those keys in  $\Sigma$  which are violated by  $T$  and maintain at the same time the violation of  $\varphi$ . Eventually, this results into a chased version of  $T$  which finitely satisfies all keys in  $\Sigma$  and violates  $\varphi$ . This would show that  $\varphi$  is not implied by  $\Sigma$ . However, the completeness proof is flawed since it requires the soundness of the *subnodes* rule, but the *subnodes* rule is not sound as discussed above. Moreover, the *subnodes-epsilon* rule is sound and independent from the inference rules of [9]. Therefore, the rules are not complete even for the original definition of XML keys [8] whose key paths are  $PL_s$  expressions.

Our technique does not use a chase: we will first represent  $\varphi \notin \Sigma^+$  in terms of a finite node-labelled tree  $T_{\Sigma, \varphi}$ , and then calculate the impact of each key in  $\Sigma$  on a finite XML tree  $T$  that satisfies  $\Sigma$ , but violates  $\varphi$ . We keep track of the impacts by inserting edges into a node-labelled digraph  $G_{\Sigma, \varphi}$ , called witness graph. Finally, we apply a reachability algorithm to  $G_{\Sigma, \varphi}$  to generate the desired XML tree  $T$ . This approach turns out to provide even more than just proving completeness. In fact,  $\Sigma$  implies  $\varphi$  if and only if there is a path from a distinguished node  $q'_\varphi$  to a distinguished node  $q_\varphi$  in  $G_{\Sigma, \varphi}$ . We will see later on that this observation results in a surprisingly efficient decision procedure for the implication problem of XML keys.

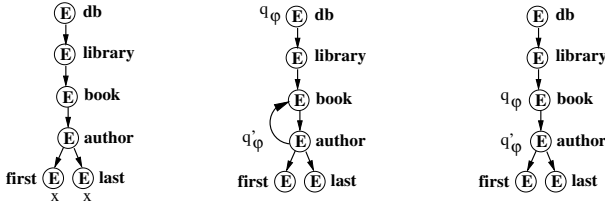
### 3.1 Mini-trees and Witness Graphs

Let  $\Sigma \cup \{\varphi\}$  be a finite set of keys in  $\mathcal{K}$ . Let  $\mathcal{L}_{\Sigma, \varphi}$  denote the set of all labels  $\ell \in \mathcal{L}$  that occur in path expressions of keys in  $\Sigma \cup \{\varphi\}$ , and fix a label  $\ell_0 \in \mathbf{E} - \mathcal{L}_{\Sigma, \varphi}$ . Further, let  $O_\varphi$  and  $O'_\varphi$  be the  $PL_s$  expressions obtained from the  $PL$  expressions  $Q_\varphi$  and  $Q'_\varphi$ , respectively, when replacing each  $_*$  by  $\ell_0$ .

Let  $p$  be an  $O_\varphi$ -path from a node  $r_\varphi$  to a node  $q_\varphi$ , let  $p'$  be an  $O'_\varphi$ -path from a node  $r'_\varphi$  to a node  $q'_\varphi$  and, for each  $i = 1, \dots, k_\varphi$ , let  $p_i$  be a  $P_i^\varphi$ -path from a node  $r_i^\varphi$  to a node  $x_i^\varphi$ , such that the paths  $p, p', p_1, \dots, p_{k_\varphi}$  are mutually node-disjoint. From the paths  $p, p', p_1, \dots, p_{k_\varphi}$  we obtain the *mini-tree*  $T_{\Sigma, \varphi}$  by identifying the node  $r'_\varphi$  with  $q_\varphi$ , and by identifying each of the nodes  $r_i^\varphi$  with  $q'_\varphi$ . Note that  $q_\varphi$  is the unique node in  $T_{\Sigma, \varphi}$  that satisfies  $q_\varphi \in \llbracket O_\varphi \rrbracket$ , and  $q'_\varphi$  is the unique node in  $T_{\Sigma, \varphi}$  that satisfies  $q'_\varphi \in q_\varphi \llbracket O'_\varphi \rrbracket$ .

In the sequel, we will discuss how to construct an XML tree from  $T_{\Sigma, \varphi}$  that could serve as a counter-example for the implication of  $\varphi$  by  $\Sigma$ . A major step in this construction is the duplication of certain nodes of  $T_{\Sigma, \varphi}$ . To begin with, we determine those nodes of  $T_{\Sigma, \varphi}$  for which we will generate two value equal copies in a possible counter-example tree. The *marking* of the mini-tree  $T_{\Sigma, \varphi}$  is a subset  $\mathcal{M}$  of the node set of  $T_{\Sigma, \varphi}$ : if for all  $i = 1, \dots, k_\varphi$  we have  $P_i^\varphi \neq \varepsilon$ , then  $\mathcal{M}$  consists of the leaves of  $T_{\Sigma, \varphi}$ , and otherwise  $\mathcal{M}$  consists of all descendant-or-selfs of  $q'_\varphi$  in  $T_{\Sigma, \varphi}$ . The nodes in  $\mathcal{M}$  are said to be *marked*.

*Example 2.* The left of Figure 3 shows the mini-tree  $T_{\Sigma, \varphi}$  for the key  $\varphi = (\varepsilon, (_*.book, (author, \{first, last\})))$  and some  $\Sigma$ , where *library* is the fixed label chosen from  $\mathbf{E} - \mathcal{L}_{\Sigma, \varphi}$ . The marking of the mini-tree consists of its leaves (emphasised by  $\times$ ).  $\square$



**Fig. 3.** A mini-tree and two witness graphs

We use mini-trees to calculate the impact of a key in  $\Sigma$  on a possible counterexample tree for the implication of  $\varphi$  by  $\Sigma$ . To distinguish keys that have an impact from those that do not, we introduce the notion of *applicability*.

**Definition 2.** Let  $T_{\Sigma, \varphi}$  be the mini-tree of the key  $\varphi$  with respect to  $\Sigma$ , and let  $\mathcal{M}$  be its marking. A key  $\sigma$  is said to be applicable to  $\varphi$  if and only if there are nodes  $w_\sigma \in \llbracket Q_\sigma \rrbracket$  and  $w'_\sigma \in w_\sigma \llbracket Q'_\sigma \rrbracket$  in  $T_{\Sigma, \varphi}$  such that  $w'_\sigma \llbracket P_i^\sigma \rrbracket \cap \mathcal{M} \neq \emptyset$  for all  $i = 1, \dots, k_\sigma$ . We say that  $w_\sigma$  and  $w'_\sigma$  witness the applicability of  $\sigma$  to  $\varphi$ .  $\square$

*Example 3.* Let  $\Sigma$  consist of the two keys  $\sigma_1 = (\varepsilon, (-*.book, \{\text{isbn}\}))$  and  $\sigma_2 = (-*.book, (\text{author}, \{\text{first}, \text{last}\}))$ , and let  $\varphi = (\varepsilon, (-*.book.author, \{\text{first}, \text{last}\}))$ . We find that  $\sigma_1$  is not applicable to  $\varphi$ , while  $\sigma_2$  is indeed applicable to  $\varphi$ .  $\square$

We define the *witness graph*  $G_{\Sigma, \varphi}$  as the node-labelled digraph obtained from  $T_{\Sigma, \varphi}$  by inserting additional edges: for each key  $\sigma \in \Sigma$  that is applicable to  $\varphi$  and for each pair of nodes  $w_\sigma \in \llbracket Q_\sigma \rrbracket$  and  $w'_\sigma \in w_\sigma \llbracket Q'_\sigma \rrbracket$  that witness the applicability of  $\sigma$  to  $\varphi$ ,  $G_{\Sigma, \varphi}$  should contain the edge  $(w'_\sigma, w_\sigma)$ . Subsequently, we refer to these additional edges as *witness edges*, while the original edges from  $T_{\Sigma, \varphi}$  are referred to as *downward edges* of  $G_{\Sigma, \varphi}$ . This is motivated by the fact that for every witness  $w_\sigma$  and  $w'_\sigma$ , the node  $w'_\sigma$  is a descendant-or-self of the node  $w_\sigma$  in  $T_{\Sigma, \varphi}$ , and thus the witness edge  $(w'_\sigma, w_\sigma)$  is an upward edge or loop in  $G_{\Sigma, \varphi}$ .

*Example 4.* Let  $\Sigma = \{\sigma_1, \sigma_2\}$  as in Example 3, and let  $\varphi$  be the key  $(\varepsilon, (-*.book.author, \{\text{first}, \text{last}\}))$ . The witness graph  $G_{\Sigma, \varphi}$  is illustrated in the middle of Figure 3. It contains a witness edge arising from  $\sigma_2$ .  $\square$

*Example 5.* Let  $\Sigma$  consist of the single key  $\sigma = (-*.book, (\text{author}, \{\varepsilon\}))$ , and let  $\varphi = (-*.book, (\text{author}, \{\text{first}, \text{last}\}))$ . The witness graph  $G_{\Sigma, \varphi}$  is illustrated in the right of Figure 3. It does not contain any witness edges since  $\sigma$  is not applicable to  $\varphi$  due to  $q'_\varphi \llbracket \varepsilon \rrbracket \cap \mathcal{M} = \emptyset$ .  $\square$

### 3.2 Reachability vs. Derivability

We show that if there is a dipath from  $q'_\varphi$  to  $q_\varphi$  in  $G_{\Sigma, \varphi}$ , then  $\varphi \in \Sigma^+$ . In other words, if  $\varphi \notin \Sigma^+$ , then there is no dipath from  $q'_\varphi$  to  $q_\varphi$  in  $G_{\Sigma, \varphi}$ .

The proof idea of the *Main Lemma* is illustrated in Figure 4. If there is a dipath  $D$  from  $q'_\varphi$  to  $q_\varphi$  in the witness graph  $G_{\Sigma, \varphi}$ , then  $D$  takes the form

illustrated in the left of Figure 4 resulting from applicable keys  $\sigma_1, \dots, \sigma_n \in \Sigma$ . For the existence of derivable keys applicable to  $\varphi$  we can assume without loss of generality that  $\Sigma = \Sigma_{\mathcal{R}}^+$ . Notice that the *context-target* rule can be applied to extend context paths by shortening target paths appropriately. Hence, there are applicable keys in  $\Sigma$  that give rise to a dipath  $D'$  from  $q'_\varphi$  to  $q_\varphi$  that takes the form illustrated in the middle of Figure 4. The inference rules from Table 2 show that there is a single key  $\sigma \in \Sigma$  that gives rise to a dipath  $D_0$  from  $q'_\varphi$  to  $q_\varphi$  in  $G_{\Sigma, \varphi}$  illustrated in the right of Figure 4.

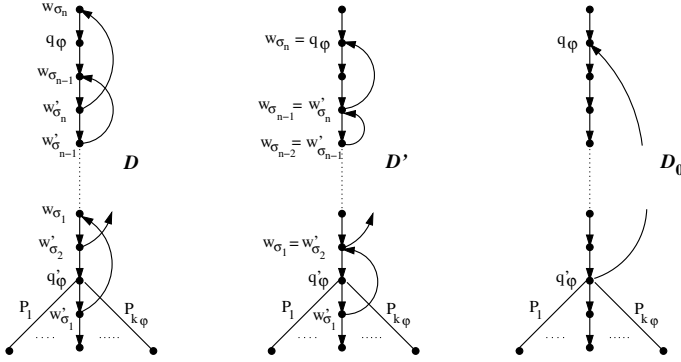


Fig. 4. Proof idea of main lemma

**Lemma 1 (Main Lemma).** *Let  $\Sigma \cup \{\varphi\}$  be a finite set of XML keys in  $\mathcal{K}$ . If  $q_\varphi$  is reachable from  $q'_\varphi$  in the witness graph  $G_{\Sigma, \varphi}$ , then  $\varphi \in \Sigma^+$ .  $\square$*

*Example 6.* In RSA the modulus  $n$  is the product of two large primes  $p$  and  $q$  which are part of the user’s private key. *Common modulus attacks* are avoided by choosing a distinct modulus  $n$  for each user, i.e., we specify the key  $(\varepsilon, (\text{group.user}, \{\text{private.p}, \text{private.q}\}))$  denoted by  $\sigma_1$ . Let  $\sigma_2$  denote  $(\text{group}, (\text{user.private}, \{p, q\}))$  stating that *private* keys can be identified by their prime numbers  $p$  and  $q$  relatively to the user’s *group*; and let  $\varphi$  denote  $(\varepsilon, (\text{group.user.private}, \{p, q\}))$  stating that there are no two private keys that contain the same primes  $p$  and  $q$ . Finally, let  $\Sigma = \{\sigma_1, \sigma_2\}$ . The mini-tree  $T_{\Sigma, \varphi}$  and witness graph  $G_{\Sigma, \varphi}$  are shown as first and second picture of Figure 5, respectively. We apply *context-target* to  $\sigma_2$  to derive  $(\text{group.user}, (\text{private}, \{p, q\}))$  denoted by  $\sigma'_2$ . Let  $\Sigma' = \{\sigma_1, \sigma'_2\}$ . The witness graph  $G_{\Sigma', \varphi}$  is shown as third picture in Figure 5. An application of the *interaction* rule to  $\sigma_1$  and  $\sigma'_2$  results in  $\varphi$  which shows that  $\varphi \in \Sigma^+$ , illustrated on the right of Figure 5.  $\square$

### 3.3 An Illustration of the Completeness Argument

Let  $\Sigma \cup \{\varphi\}$  be a finite set of keys in  $\mathcal{K}$  such that  $\varphi \notin \Sigma^+$ . In order to show completeness one needs to demonstrate that  $\varphi \notin \Sigma^*$ . Indeed, we can construct a finite XML tree  $T$  which satisfies  $\Sigma$ , but violates  $\varphi$ . Since  $\varphi \notin \Sigma^+$  we know by Lemma 1 that  $q_\varphi$  is not reachable from  $q'_\varphi$  in  $G_{\Sigma, \varphi}$ . Let  $u$  denote the bottom-most

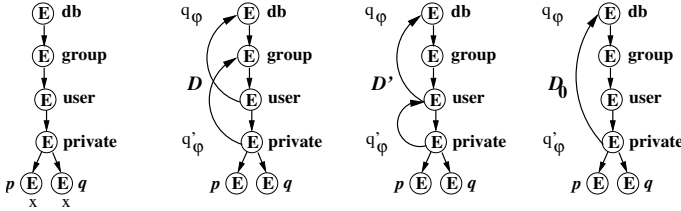


Fig. 5. Example of Main Lemma

descendant-or-self of  $q_\varphi$  in  $T_{\Sigma,\varphi}$  such that  $q_\varphi$  is still reachable from  $u$  in  $G_{\Sigma,\varphi}$ . Further,  $u$  must be a proper ancestor of  $q'_\varphi$  because otherwise  $u$  and thus  $q_\varphi$  were reachable from  $q'_\varphi$  in  $G_{\Sigma,\varphi}$ . Let  $T_0$  denote a copy of the path from  $r$  to  $u$ , and  $T_1, T_2$  denote two node-disjoint copies of the subtree of  $T_{\Sigma,\varphi}$  rooted at  $u$ .  $T_1$  and  $T_2$  are populated with text-leaves such that a node of  $T_1$  and a node of  $T_2$  become value-equal precisely when they are copies of the same marked node in  $T_{\Sigma,\varphi}$ . The counter-example tree  $T$  is obtained from  $T_0, T_1, T_2$  by identifying the terminal node of  $T_0$  with the roots of  $T_1$  and  $T_2$ .

The left side of Figure 6 shows a counter-example tree  $T$  for the implication of  $\varphi$  by  $\Sigma = \{\sigma_1, \sigma_2\}$  from Example 4. In this case,  $u$  is the single *db*-node in  $T_{\Sigma,\varphi}$ , and  $T_1, T_2$  are two node-disjoint copies of the entire tree  $T_{\Sigma,\varphi}$ . Furthermore,  $T_1$  and  $T_2$  carry the same text under their *first*-nodes and their *last*-nodes, respectively, and carry different text anywhere else.  $T$  itself is obtained by identifying the three copies of the *db*-node from  $T_0, T_1$  and  $T_2$ .

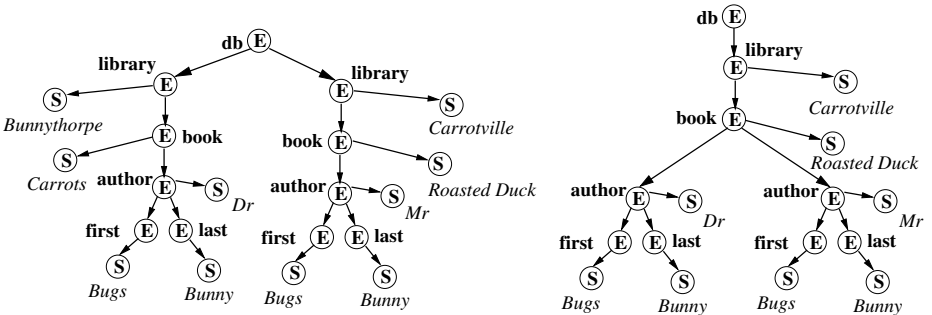


Fig. 6. Counter-example trees for the implication of  $\varphi$  by  $\Sigma = \{\sigma_1, \sigma_2\}$  from Example 4, and for the implication of  $\varphi$  by  $\sigma$  in Example 5

## 4 Deciding Implication

We will show in this section how our technique of proving completeness can be applied to obtain an algorithm for deciding XML key implication in time quadratic in the size of the constraints. Notice that the heuristic time algorithm from [9] is flawed since it is based on the soundness of the subnodes rule. Even if only  $PL_s$  expressions are considered for the key paths, the algorithm is still flawed since it is based on the incomplete set  $\mathfrak{S}$  of inference rules.

## 4.1 The Algorithm

The first step is to show that XML key implication can be completely characterised in terms of the reachability problem of nodes in the witness graph.

**Theorem 2.** *Let  $\Sigma \cup \{\varphi\}$  be a finite set of keys in  $\mathcal{K}$ . We have  $\Sigma \models \varphi$  if and only if  $q_\varphi$  is reachable from  $q'_\varphi$  in  $G_{\Sigma, \varphi}$ .*  $\square$

Theorem 2 suggests to utilise the following algorithm for deciding XML key implication. Our technique for proving the completeness of our inference rules results in a very compact and easily comprehensible algorithm.

### Algorithm 1 (XML Key-Implication)

**Input:** finite set  $\Sigma \cup \{\varphi\}$  of XML keys

**Output:** yes, if  $\Sigma \models \varphi$ ; no, if  $\Sigma \not\models \varphi$

**Method:**

- (1) Construct  $G_{\Sigma, \varphi}$  from  $\Sigma$  and  $\varphi$ ;
- (2) **IF**  $q_\varphi$  is reachable from  $q'_\varphi$  in  $G_{\Sigma, \varphi}$  **THEN RETURN**(yes)
- (3) **ELSE RETURN**(no).

The correctness of Algorithm 1 is an immediate consequence of Theorem 2.

## 4.2 The Time Complexity

We analyse the time complexity of Algorithm 1. The problem of deciding whether  $q_\varphi$  is reachable from  $q'_\varphi$  in  $G_{\Sigma, \varphi}$  can be solved by applying a depth-first search algorithm to  $G_{\Sigma, \varphi}$  with root  $q'_\varphi$ . This algorithm works in time linear in the number of edges of  $G_{\Sigma, \varphi}$  [23]. Since the number of nodes in  $G_{\Sigma, \varphi}$  is just  $|\varphi| + 1$ , step (2) of Algorithm 1 can be executed in  $\mathcal{O}(|\varphi|^2)$  time. It therefore remains to investigate the time complexity for generating the witness graph  $G_{\Sigma, \varphi}$  from  $\Sigma$  and  $\varphi$ . This can be done as follows:

1. Initialise  $G_{\Sigma, \varphi}$  with  $T_{\Sigma, \varphi}$ ;
2. For all  $\sigma \in \Sigma$ , add the edge  $(w'_\sigma, w_\sigma)$  to  $G_{\Sigma, \varphi}$  whenever  $w_\sigma$  and  $w'_\sigma$  witness the applicability of  $\sigma$  to  $\varphi$  (and the edge does not already exist).

A semi-naive way to execute the last of these steps is to evaluate  $w'_\sigma \llbracket P_i^\sigma \rrbracket$  for  $i = 1, \dots, k_\sigma$ , for all  $w'_\sigma \in w_\sigma \llbracket Q'_\sigma \rrbracket$  and all  $w_\sigma \in \llbracket Q_\sigma \rrbracket$ . Recall that a query of the form  $v \llbracket Q \rrbracket$  is a *Core XPath* query and can be evaluated on a node-labelled tree  $T$  in  $\mathcal{O}(|T| \times |Q|)$  time [18]. Hence, we can evaluate  $w'_\sigma \llbracket P_i^\sigma \rrbracket$  for all  $i = 1, \dots, k_\sigma$  in time  $\mathcal{O}(|\varphi| \times |\sigma|)$ . Since  $\llbracket Q_\sigma \rrbracket$  and  $w_\sigma \llbracket Q'_\sigma \rrbracket$  contain at most  $|\varphi|$  nodes each, this step can be executed in  $\mathcal{O}(|\varphi|^3 \times |\sigma|)$  time for each  $\sigma$ . If  $\|\Sigma\|$  denotes the sum of all sizes  $|\sigma|$  for  $\sigma \in \Sigma$ , then we need  $\mathcal{O}(\|\Sigma\| \times |\varphi|^3)$  time to generate  $G_{\Sigma, \varphi}$ .

Using a more involved analysis of witness edges, we can show that  $q_\varphi$  is reachable from  $q'_\varphi$  in  $G_{\Sigma, \varphi}$  if and only if  $q_\varphi$  is reachable from  $q'_\varphi$  in  $H_{\Sigma, \varphi}$  - where  $H_{\Sigma, \varphi}$  is a subgraph of  $G_{\Sigma, \varphi}$  in which certain witness edges are omitted. Moreover,  $H_{\Sigma, \varphi}$  can be computed in time  $\mathcal{O}(\|\Sigma\| \times |\varphi|)$ . Hence, we obtain:

**Theorem 3.** *Let  $\Sigma \cup \{\varphi\}$  be a finite set of XML keys in  $\mathcal{K}$ . The implication problem  $\Sigma \models \varphi$  can be decided in  $\mathcal{O}(|\varphi| \times (\|\Sigma\| + |\varphi|))$  time.*  $\square$

## 5 Future Work

One area that warrants future research is the study of keys with respect to more expressive path languages [5,13,25,26,34]. In particular, the problems of axiomatising and developing efficient algorithms that decide the implication of XML keys as defined in [9] are still open.

A different way of increasing the expressiveness is to view keys from a different angle. Indeed, keys restrict the number of nodes, which have the same value on certain selected subnodes, to 1. It is therefore natural to introduce numerical keys that simply restrict the number of nodes, having the same value on certain selected subnodes, to an arbitrary finite number. We can axiomatise this class of XML constraints and provide practically efficient algorithms to reason about them. Due to lack of space we omit details. Efficient reasoning about such constraints allows to precompute upper bounds on the number of query answers.

## 6 Conclusion

We have reviewed the key constraint language for XML [8,9] and observed that their axiomatisation [9] is not sound for XML key implication in general [9] and not complete even for XML keys with simple key path expressions [8]. Based on this observation we have provided an axiomatisation of XML keys as defined in [8]. Our technique allows us to characterise XML key implication in terms of the reachability problem for nodes in a digraph. This results in a first correct decision procedure, that is also practically efficient, i.e., quadratic in the size of the input. Keys form a very natural class of XML constraints that can be utilised effectively by designers, and the complexity of their associated decision problems indicates that they can be maintained efficiently by database systems for XML applications.

## References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. V. Apparao et al. Document object model (DOM) level 1 specification, W3C recommendation, oct. 1998. <http://www.w3.org/TR/REC-DOM-Level-1/>.
3. M. Arenas and L. Libkin. A normal form for XML documents. *TODS*, 29(1):195–232, 2004.
4. M. Arenas and L. Libkin. An information-theoretic approach to normal forms for relational and XML data. *J. ACM*, 52(2):246–283, 2005.
5. M. Benedikt, W. Fan, and G. Kuper. Structural properties of XPath fragments. *TCS*, 336(1):3–31, 2005.
6. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0 (third edition) W3C recommendation, feb. 2004. <http://www.w3.org/TR/2004/REC-xml-20040204/>.
7. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Reasoning about keys for XML. In *DBPL*, pages 133–148, 2001.

8. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. *Computer Networks*, 39(5):473–487, 2002.
9. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Reasoning about keys for XML. *Inf. Syst.*, 28(8):1037–1063, 2003.
10. P. Buneman, W. Fan, J. Siméon, and S. Weinstein. Constraints for semi-structured data and XML. *SIGMOD Record*, 30(1):47–54, 2001.
11. P. Buneman, W. Fan, and S. Weinstein. Path constraints in semistructured databases. *JCSS*, 61(2):146–193, 2000.
12. J. Clark and S. DeRose. XML path language (XPath) version 1.0, W3C recommendation, nov. 1999. <http://www.w3.org/TR/xpath>.
13. A. Deutsch and V. Tannen. Containment and integrity constraints for XPath. In *KRDB*, 2001.
14. R. Fagin and M. Y. Vardi. The theory of data dependencies. In *ICALP*, pages 1–22, 1984.
15. W. Fan. XML constraints. In *DEXA Workshops*, pages 805–809, 2005.
16. W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM*, 49(3):368–406, 2002.
17. W. Fan and J. Siméon. Integrity constraints for XML. *JCSS*, 66(1):254–291, 2003.
18. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. *TODS*, 30(2):444–491, 2005.
19. C. Hara and S. Davidson. Reasoning about nested functional dependencies. In *PODS*, pages 91–100, 1999.
20. S. Hartmann and S. Link. More functional dependencies for XML. In *ADBIS*, number 2798 in LNCS, pages 355–369. Springer, 2003.
21. S. Hartmann and S. Link. Multivalued dependencies in the presence of lists. In *PODS*, pages 330–341, 2004.
22. S. Hartmann and T. Trinh. Axiomatizing functional dependencies for XML with frequencies. In *FoIKS*, number 3861 in LNCS, pages 159–178. Springer, 2006.
23. D. Jungnickel. *Graphs, Networks and Algorithms*. Springer, 1999.
24. M. Kay. XSL transformations (XSLT) version 2.0 W3C candidate recommendation, nov. 2005. <http://www.w3.org/TR/xslt20/>.
25. G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004.
26. F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *ICDT*, pages 315–329, 2003.
27. J. Paredaens, P. De Bra, M. Gyssens, and D. Van Gucht. *The Structure of the Relational Database Model*. Springer, 1989.
28. D. Suciu. On database theory and XML. *SIGMOD Record*, 30(3):39–45, 2001.
29. B. Thalheim. *Dependencies in Relational Databases*. Teubner, 1991.
30. H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures Second Edition, W3C Recommendation, 28 Oct. 2004. <http://www.w3.org/TR/xmlschema-1/>.
31. V. Vianu. A web odyssey: from Codd to XML. *SIGMOD Record*, 32(2):68–77, 2003.
32. M. Vincent, J. Liu, and C. Liu. Strong functional dependencies and their application to normal forms in XML. *TODS*, 29(3):445–462, 2004.
33. J. Widom. Data management for XML: Research directions. *Data Eng. Bull.*, 22(3):44–52, 1999.
34. P. Wood. Containment for XPath fragments under DTD constraints. In *ICDT*, pages 300–314, 2003.