Validation of Expressive XML Keys with XML Schema and XQuery

Bo Liu[1] **Sebastian Link**[1] **Emir Muñoz**[2]

[1] Department of Computer Science,
The University of Auckland, New Zealand
Email: [s.link|b.liu]@auckland.ac.nz

[2] Insight Centre for Data Analytics,
National University of Ireland Galway, Ireland
Email: emir.munoz@insight-centre.org

Abstract

The eXtensible Markup Language (XML) is the de-facto industry standard for exchanging data on the Web and elsewhere. While the relational model of data enjoys a well-accepted definition of a key, several competing notions of keys exist in XML. These have complementary properties and therefore serve different applications domains. In a nutshell, XML keys allow us to capture important domain semantics in XML documents and thereby advance data processing in most applications. In this paper we propose how to validate XML documents against an expressive class of XML keys using XML Schema and XQuery, respectively. It is somewhat surprising how simple it is to express sophisticated notions of XML keys in these off-the-shelf tools. Experiments show that our simple validation technique works well for real-world data of reasonable size. For large-scale data, however, dedicated tools must be developed.

Keywords: Key, Performance, Schema, Semantics, Query, Validation, XML

1 Introduction

The eXtensible Markup Language (XML) (Bray et al. 2006) has evolved to become the de-facto industry standard for the sharing and integration of data. This is mainly due to the syntactic flexibility by which end users can produce XML documents. Unfortunately, this flexibility is an inhibitor when it comes to checking XML documents for their validity with respect to application semantics. XML schema languages such as document type definitions and the standard XML Schema provide rich constructs to impose many structural constraints (Abiteboul et al. 2000, Arenas & Libkin 2004, 2005, Buneman et al. 2000, Buneman, Fan, Siméon & Weinstein 2001, Buneman et al. 2002, Deutsch & Tannen 2005, Fan & Siméon 2003, Fan 2005, Hartmann & Link 2003, 2009, Vianu 2003, Vincent et al. 2004, 2007), but have significant shortcomings when it comes to semantics. Keys, as proposed by XML Schema, are not naturally defined and show provably bad computational properties (Arenas et al. 2002, 2008). However, the importance of keys for XML has been recognized by industry and academia. Keys provide the means for identifying data items in an unambiguous way, which is essential for retrieving and updating data. More importantly, unlike relational data (Abiteboul et al. 1995, Thalheim 1991) in which there is a unique and well-accepted concept of a key, for XML data (Hartmann et al. 2007) there are many different ways of defining the semantics of XML keys, each with its own advantages and disadvantages. Therefore, over the past few years, the research literature has proposed alternative key languages (Buneman et al. 2002, Hartmann & Link 2007, Hartmann et al. 2007, Karlinger et al. 2009) which are reminiscent of keys from databases, do not require an XML Schema definition, and have provably good computational properties (Hartmann & Link 2007, 2009).

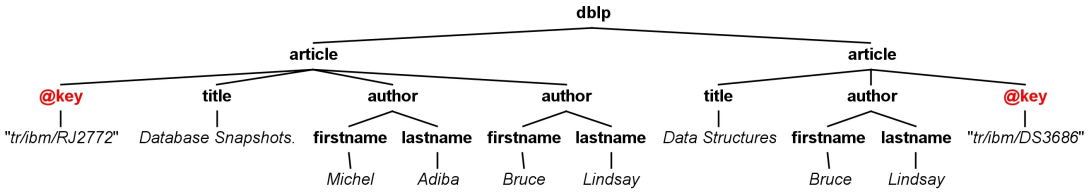
Figure 1 shows an XML data tree which represents two article nodes with their sub-nodes. The *key* value identifies the article node, because the *key* sub-nodes of the different article nodes have different values. In contrast, an article node cannot be identified in the entire tree by its child node author, because the same author has written more than one article. However, the article node can indeed be identified by its child author node together with its child title node. That means, for each article node, different child nodes of article must differ on their author or title sub-node value. The XML keys in this simple example have a different expressivity than those proposed in XSDs or DTDs (Hartmann et al. 2007, Bray et al. 2006, Thompson et al. 2004).

(Buneman et al. 2002) have introduced an alternative language to specify XML keys, which has a natural semantics, does not require an XML Schema definition and exhibits excellent computational properties (Hartmann & Link 2009). For example, the semantics of the keys above can be specified as

1. (.,/article,{key}),
2. (.,/article,{author}), and
3. (.,/article,{author,title})

The main body of research work on XML keys has been devoted to the computational properties of their associated consistency and implication problems (Arenas et al. 2008, Buneman et al. 2003, Hartmann & Link 2009, 2010, Karlinger et al. 2009), the important problem of validating XML keys has received only little attention so far (Chen et al. 2002, Liu et al. 2004, 2005). The validation problem for a class of XML keys decides whether a given XML document satisfies a given XML key from that class. For example, the XML data tree from Figure 1 is valid with respect to the first and with respect to the third key above, but invalid with respect to the second key. The validation problem is important for a number of reasons. In fact, it allows us to validate whether a given XML document respects the semantics of its application domain encoded by a given XML key. This is the

Figure 1: XML data tree



basis for modeling application data correctly, ensuring higher levels of data quality, and enabling other applications to process data more efficiently. For organizations this might mean an increase in productivity, less resource requirements on data cleaning, and improvements for data-driven decision making. However, XML key validation poses a challenge since the semantics of the XML keys, as proposed originally by (Buneman et al. 2002), is expressive, and it is therefore not obvious how to propose good practical solutions to the validation problem.

1.1 Contribution

Our research makes two main contributions. Firstly, it is shown how the validation problem for XML keys as proposed by (Buneman et al. 2002) can be decided by evaluating an XQuery on the given document. For this, the semantics of the XML keys is encoded as a simple conditional statement in XQuery that takes advantage of nested quantified expressions such as `SATISFIES`, `SOME`, and `ALL`, as well as the `DEEP-EQUAL` function that essentially tests whether two given nodes of an XML data tree form the roots of isomorphic trees. It is stressed that our XQuery condition can also express XML keys that were not included in the original proposal by (Buneman et al. 2002) and any other follow-up work. Indeed, the XQuery condition allows us to exploit any XPath axes in the semantics and validation of XML keys. It is further demonstrated how to decide the validation problem for this class of XML keys with the help of an assertion in XML Schema 1.1. Our first contribution therefore enables organizations to validate their documents against expressive application semantics by off-the-shelf tools. This is a somewhat surprisingly simple and easy-to-use solution.

Secondly, we investigate how well our easy-to-use validation method with XQuery performs on real-world data sets. For this purpose, we conduct performance tests on fragments of an astronomical data set with growing sizes of up to 23MB, as well as the DBLP data set with growing sizes of up to 127MB. The experiments are applied to several fragments of XML keys that exploit different XPath axes. The experiments show that our validation method works efficiently on modestly-sized XML documents, while larger XML documents require a long time to be validated against simple XML keys. This is a consequence of directly encoding the original nested quantifier semantics of XML keys in the XQuery condition. It is future research to trade-in the simplicity of this XQuery condition against designated validation implementations that scale better to larger data sets.

1.2 Organization

The remainder of the paper is organized as follows. The small body of previous research on XML key validation is reviewed in Section 2. The XML data model is defined in Section 3. The keys proposed by XML Schema are discussed in Section 4, and Section 5 provides an overview of proposals from the academic community. In Section 6, we present our proposal of using XQuery to validate XML documents against the class of XML keys from (Buneman et al. 2002). The performance evaluation of our proposal is presented in Section 7. An example illustrates in Section 8 how assertions of XML Schema 1.1 can be used to validate XML documents against XML keys.

2 Related Work

Previous work on XML keys is discussed in detail in Sections 4 and 5, where we review the W3C recommendation for XML keys as well as various proposals from academia. In this section, we only focus on the validation problem of XML keys.

All previous work on XML key validation has developed designated algorithms. These are therefore different from our proposal, which simply encodes the semantics of an expressive class of XML keys (Buneman et al. 2002) within XQuery and XML Schema 1.1, respectively. In contrast to all other works, our proposal also supports XPath axes other than the ones used by (Buneman et al. 2002).

(Chen et al. 2002) present an XML key constraint validator based on SAX. Their tool, called *XKvalidator* also considers the class of XML keys from (Buneman et al. 2002), but does not support further XPath axes. The validator can be used both for bulk-loading, i.e. one pass over the entire document, as well as for incremental checking, i.e. XML updates to the document can be processed and checked against a persistent key index for the file.

(Abrão et al. 2004) and (Bouchou et al. 2003) also consider the keys of (Buneman et al. 2002). They introduce a method for building an XML constraint validator from a given set of schema, key and foreign key constraints. The XML constraint validator obtained by their method is a bottom-up tree transducer that is used not only for one-pass checking of the correctness of an XML document but also for incrementally validating updates over this document. In this way, both the verification from scratch and the update verification are based on regular (finite and tree) automata, making the whole process efficient.

Since the XML keys themselves are expressed in XPath and the structure checking is a well-solved problem, the validation using XPath based on DOM and the structure checking is a principal motivation for the work by (Liu et al. 2004, 2005). They propose an algorithm that generates a key value document and present how such a document and its schema can be designed to check whether predefined key constraints are satisfied. They present a method that supports the incremental validation of XML keys by the incremental maintenance of their key value document.

3 XML Data Model

Researchers and practitioners agree that in terms of storing, querying and processing XML data in its native format, XML requires commensurate support by DBMS and data management tools. It is a challenge to precisely describe desirable properties of XML data and provide methods and facilities that can efficiently conclude, validate, or enforce such properties, because XML has an inherent syntactic flexibility and hierarchical structure (Fan 2005, Fan & Libkin 2002, Fan & Siméon 2003, Suciu 2001, Vianu 2003).

Integrity constraints restrict data stores such as XML documents or XML databases to those considered meaningful for some application of interest. Specifying and enforcing integrity constraints helps to ensure that the data stays valid and does not deviate from reality (Fan 2005). Keys are one of the most fundamental classes of integrity constraints. The importance of keys for XML has been recognized by industry and academia. Keys provide the means for identifying data items in an unambiguous way, an ability that is essential for retrieving and updating data. For relational data, keys are straightforward to define, convenient to use and simple to reason about. For XML data however, the story is more cumbersome due to the particularities of the XML data model. Over the past few years several notions have been proposed and discussed in the research community, including the definitions that have been introduced into XML Schema (Thompson et al. 2004). While this has established an industry standard for specifying keys, (Arenas et al. 2002) have shown the computational intractability of the associated consistency problem, that is, whether there exists an XML document that conforms to a given DTD or XSD and satisfies the specified keys.

In Figure 1, an example of a reasonable key is that the *key*-value identifies the *article* node. That is, the *key* subnodes of different *article* nodes must have different values. In contrast, an *author* cannot be identified in the entire tree by its *firstname* and *lastname* subnodes since the same *author* can write more than one *article*. However, the *author* can indeed be identified by its *firstname* and *lastname* subnodes in relation to the *article* node. That is, for each individual *article* node, different *author* subnodes must differ on their *firstname* or *lastname* subnode value. Clearly, the expressiveness of such keys depends on the means that are used to select nodes in a tree, and by the semantics of the associated node selection queries (Fan 2005, Fan & Libkin 2002, Fan & Siméon 2003, Suciu 2001, Vianu 2003).

3.1 Trees and Paths in XML

It is common to represent XML data by ordered, node-labelled trees. Such a representation is used in DOM (Apparao et al 1998), XPath (Clark & DeRose 1999), XQuery (Chamberlin et al. 2010), XSLT (Kay 2009), and XML Schema (Thompson et al. 2004). Figure 1 shows an example of XML data represented as a tree. Non-leaves are always of type E for element, while leaves are either of type A for attribute (when they have a name starting with @ and a string underneath) or of type S for string (PCDATA). In the literature, several variations of the tree model for XML data have been used. All of them simplify the XML standard (Bray et al. 2006) by concentrating on its major aspects. Here we follow the approach taken in (Buneman et al. 2002, 2003). We assume that there are three mutually disjoint non-empty sets **E**, **A**, and **S** = {S}. In the sequel, **E** will be used for element names, **A** for attribute names, and **S** for denoting text. We further assume that these sets are pairwise disjoint, and put $\mathcal{L} = \mathbf{E} \cup \mathbf{A} \cup \mathbf{S}$. 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 $V \to \mathcal{L}$ assigning a label to every node in V ; a node v in V is called an element (E) node if $lab(v) \in E$, an attribute (A) node if $lab(v) \in A$, and a text (S) node if $lab(v) = \mathrm{S}$; *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; 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. Let E_T denote the set of edges of T, and let E_T^* denote the transitive closure of E_T. *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 then r is the unique and distinguished root node of T.

A path expression is a finite sequence of zero or more symbols from some alphabet **M**. The unique sequence of zero symbols is called the empty path expression,denoted by ε, and a dot (.) is used to denote the concatenation of path expressions. The set of all path expressions over **M**, with the binary operation of concatenation and the identity ε, form a free monoid. We are in particular interested in path expressions over the alphabet $\mathcal{L}$ which we call simple. A simple path p of an XML tree T is a sequence of nodes $v_0, \ldots, v_m$ where (v_{i-1}, v_i) is an edge for $i = 1, \ldots, m$. We call p a simple path from v_0 to v_m, and say that v_m is reachable from v_0 following the simple path p. The path p gives rise to a simple path expression $lab(v_1), \ldots, lab(v_m)$, which we denote by $lab(p)$. An XML tree T has a tree structure: for each node v of T, there is a unique simple path from the root r to v (Hartmann et al. 2007).

3.2 Value Equality of XML Nodes

Equality is essential to the definition of keys, in other words, equality of the values associated with nodes is crucial to define the semantics of keys. In general, two nodes u and v are value equal in an XML tree T, if they have the same label. Moreover, if two nodes are string nodes, attribute nodes, or element nodes, they are value equal if they have the same string value or their children are pairwise value equal. More formally, two nodes $u, v \in V$ are value equal, denoted by $u =_v v$, if and only if the subtrees rooted at u and v are isomorphic by an isomorphism that is the identity on string values. Therefore, two nodes u and v are value equal (Buneman et al. 2002, Hartmann & Link 2009), if:

1. $lab(u) = lab(v)$,
2. if u, v are attribute node or string nodes, then $val(u) = val(v)$,
3. if u, v are element nodes, then (i) if $att(u) = \{a_1, \ldots, a_m\}$, then $att(v) = \{a'_1, \ldots, a'_m\}$ and there is a permutation π on $\{1, \ldots, m\}$ such that $a_i =_v a'_{\pi(i)}$, for $i = 1, \ldots, m$, and (ii) if $ele(u) = [u_1, \ldots, u_k]$, then $ele(v) = [v_1, \ldots, v_k]$ and $u_i =_v v_i$ for $i = 1, \ldots, k$.

If we look at Figure 1, the second and third author node (based on the document order) are value equal.

Note that the notion of value equality takes the document order of the XML tree into account. We remark that $=_v$ defines an equivalence relation on the node set of an XML tree.

To summarize, assuming there are two subsets U and W of V. We call U and W value-equal if there exists a bijection $\beta : U \to W$ such that $u =_v \beta(u)$ for all $u \in U$. The value intersection $U \cap_v W$ of U and W consists of all pairs $(u, w) \in U \times W$ such that $u =_v w$ holds, and the value difference $U -_v W$ consists of all nodes in U that are not value equal to any node in W (Hartmann et al. 2007).

3.3 Queries to Select XML Nodes

A node selection query Q defines a mapping $[\![Q]\!]_T : V \to 2^V$ that assigns every node $v \in V$ a subset of V, called the selected nodes, that may be seen as the result of executing the query at node v. For node selection queries we can define operations like union, intersection, concatenation, reverse, absolution and the identity as follows, cf. (Clark & DeRose 1999):

$$\begin{aligned}
[\![Q_1 \cup Q_2]\!]_T(v) &:= [\![Q_1]\!]_T(v) \cup [\![Q_2]\!]_T(v) \\
[\![Q_1 \cap Q_2]\!]_T(v) &:= [\![Q_1]\!]_T(v) \cap [\![Q_2]\!]_T(v) \\
[\![Q_1.Q_2]\!]_T(v) &:= \{x : w \in [\![Q_1]\!]_T(v), x \in [\![Q_2]\!]_T(w)\} \\
[\![Q^R]\!]_T(v) &:= \{x : v \in [\![Q]\!]_T(x)\} \\
[\![Q^A]\!]_T(v) &:= [\![Q]\!]_T(r) \\
[\![\varepsilon]\!]_T(v) &:= \{v\}
\end{aligned}$$

Depending on the particular application, one aims to identify query languages that enable users to retrieve as much data as possible that is relevant for the underlying application domain, yet sufficiently simple to process the queries efficiently. A problem that has been widely studied in the literature due to its immediate practical relevance is the containment problem. A node selection query Q is said to be contained in a node selection query Q', denoted by $Q \sqsubseteq Q'$, if for every XML tree T and every node $v \in V$ we have that $[\![Q]\!]_T(v)$ is a subset of $[\![Q']\!]_T(v)$. Two queries are (semantically) equivalent, denoted by $Q \equiv Q'$, if they contain one another. The containment problem for a class of queries asks to decide containment, while the equivalence problem asks to decide equivalence for the query class under inspection.

It remains to find a convenient way to express useful node selection queries that can be evaluated efficiently. In the literature regular languages of path expressions have been widely used for this purpose. Alternatively, XPath expressions (Clark & DeRose 1999) are of course popular, too. For recent tractability results on the containment and equivalence problem for regular path languages and XPath fragments we refer to (Benedikt et al. 2005, Deutsch & Tannen 2005, Gottlob et al. 2005, Miklau & Suciu 2004, Neven & Schwentick 2006, Wood 2003).

4 XML Keys

In this section we set out a framework that compares different notions of XML keys. We then discuss the XML key proposal by XML Schema.

Keys are an essential part of database design: they are fundamental to data models and conceptual design; they provide the means by which one tuple in a relational database may refer to another tuple; and they are important in update, for they enable us to guarantee that an update will affect precisely one tuple. More philosophically, if we think of a tuple as representing some real-world entity, the key provides an invariant connection between the tuple and the entity (Abiteboul et al. 1995, Thalheim 2000).

There are two elements in defining a key: one is a set, which is the set of tuples identified by a relation name in relational databases; the other is the 'attributes', which is a collection of column names in relational terminology, which uniquely identify elements in the set.

Therefore, an XML *key* is a triple $\sigma = (C, Q, \mathcal{F})$ where C and Q are node selection queries, and $\mathcal{F} = \{F_1, \ldots, F_k\}$ is a finite set of node selection queries. Adapting the terminology of (Thompson et al. 2004), C is called the *context*, Q the *selector* and $F_1, \ldots, F_k$ the *fields* of the key σ. Given an XML tree T, $[\![C]\!]_T(r)$ is called the *context node* set. For any context node u, $[\![Q]\!]_T(u)$ is called the *target node* set, and for any target node v and every $i = 1, \ldots, k$ we call $[\![F_i]\!]_T(v)$ a *key node* set. For example, in the XML key:

(article, author, {firstname, lastname})

article is the context, author is the target, and firstname and lastname are the fields of the key. An XML tree T satisfies an XML key $\sigma = (C, Q, \mathcal{F})$ if for any target nodes u, v that belong to the same target node set it holds that if for all $i = 1, \ldots, k$ their key node sets $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ agree, then the nodes u and v themselves agree (Hartmann et al. 2007).

4.1 Notions of Node Agreement

Node agreement can be classified into the following criteria (Hartmann et al. 2007). Those criteria are based on different proposals for defining agreement of target nodes and of key node sets. For the agreement of two key node sets:

(Ka) $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ are equal,

(Kb) $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ have non-empty intersection,

(Kc) $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ are value equal,

(Kd) $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ have non-empty value intersection.

For the agreement of two target nodes u and v:

(Ta) $u = v$, that is, u and v are identical nodes

(Tb) $u =_v v$, that is, u and v are value equal nodes.

Moreover, one might want to combine these requirements with one or more of the following criteria for some or all $i = 1, \ldots, k$:

(Tc) both $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ are non-empty,

(Td) both $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ contain at most one node,

(Te) both $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ contain only attribute or text nodes.

4.2 Strong Keys in XML Schema

We treat keys as defined by XML Schema (Thompson et al. 2004). For such a key σ to hold it is necessary that for each target node v each of the key fields F_i with $i = 1, \ldots, k$ selects exactly one key node. This prerequisite calls for uniqueness and existence. Buneman et. al. (Buneman, Davidson, Fan, Hara & Tan

2001, Buneman et al. 2003) call a key with such a prerequisite strong.

XML Schema (Thompson et al. 2004) uses criterion (Kd) above for the agreement of key node sets. Note that if the prerequisite holds then criteria (Kc) and (Kd) coincide, and so do criteria (Ka) and (Kb). Moreover, this prerequisite imposes a condition on each target node. Even if there is only a single target node v in an XML tree T the key will be violated when one of the key node sets $[\![F_i]\!]_T(v)$ is not a singleton set. To avoid that in such a case the target node v agrees with itself one chooses criteria (Ta,Tc,Td) for the agreement of target nodes. Further, criterion (Te) accounts for the restriction of value equality to string equality in XML Schema. XML Schema further defines a weaker version of the previous key notion, called unique. It requires that the keys' paths exist and are unique; that is, $n[\![P_i]\!]$ contains exactly one node for $1 \leq i \leq n$. The key paths constrain the target set as follows: Take any two nodes $(n_1, n_2) \in [\![Q]\!]$ and consider the pairs of nodes found by following a key path P_i from n_1 and n_2. If all such pairs of nodes are value-equal, then the nodes n_1 and n_2 represent the same node (Hartmann et al. 2007).

4.3 Absolute and Relative Keys

A key of the form $(\varepsilon, Q, \mathcal{F})$ is named *absolute*. This means the context node is always the root node. In contrast, if the context node is any other node, then we treat it as *relative* key. It should be noted that every relative key can be converted to an absolute key by transforming the key context C into an additional key field with node agreement criterion (Kb) (Hartmann et al. 2007) above: It can be shown that an XML tree satisfies $(C, Q, \mathcal{F})$ if and only if it satisfies $(\varepsilon, C.Q, \mathcal{F} \cup \{C^A \cap Q^R\})$. However, the latter representation of the key might be less intuitive in many cases (Hartmann et al. 2007).

5 Alternative Notions for XML Keys

In this section we discuss different proposal of XML keys from the research literature. It is important to note that there are several different notions, each accommodating a different expressivity. This is different from the relational model of data where the notion of a key is well-accepted. The variety of XML keys addresses different application semantics and domains.

5.1 Keys proposed by Buneman et al.

The key notion proposed by (Buneman et al. 2002) is flexible in the choice of an appropriate query language (Hartmann et al. 2007). (Buneman et al. 2002) study and examine the absolute and relative keys that do not have the uniqueness and existence prerequisite of the strong key definition in XML Schema (Thompson et al. 2004). The reason is that strong keys are not always finitely satisfiable. That is, there are strong keys for which there is no finite XML tree to satisfy them. Based on the above assumption, (Buneman et al. 2002) define keys using criteria (Kd) for the agreement of key node sets, and only (Ta) for the agreement of target nodes.

For node selection queries they define the path language PE that consists of all path expressions over the alphabet $\mathcal{L} \cup \{\_, \_^*\}$, with the binary operation of concatenation and the empty path expression ε as identity. Here, $\_$ and $\_^*$ are symbols not in $\mathcal{L}$ that serve as the single symbol wild-card and the variable length wild-card. The semantics of path expressions from PE is defined by:

$$\begin{aligned} [\![\ell]\!]_T(v) &:= \{w : (v,w) \in E_T, lab_T(w) = \ell\} \\ [\![\_]\!]_T(v) &:= \{w : (v,w) \in E_T\} \\ [\![\_^*]\!]_T(v) &:= \{w : (v,w) \in E_T^*\} \end{aligned}$$

The semantics of the concatenation operator and of ε is defined as for general node selection queries. When comparing PE and XPath, one observes the equivalences $\varepsilon \equiv .$ and $\_ \equiv *$ and $\_^* \equiv .//.$ and $Q_1.Q_2 \equiv Q_1/Q_2$ and $Q_1//Q_2 \equiv Q_1.\_^*.Q_2$ showing that PE corresponds to the XPath fragment $XP(.,/,*,//)$. In (Buneman et al. 2002, Hartmann et al. 2007) PE expressions are used for the context C and the key selector Q, while simple path expressions are used for the key fields $F_1, \ldots, F_k$. At the end of (Buneman et al. 2002, Hartmann et al. 2007) the use of PE expressions for the key fields is briefly discussed based on a more restrictive definition of value intersection for key node sets: The limited value intersection

$$[\![F_i]\!]_T(u) \cap_{lv} [\![F_i]\!]_T$$

of key node sets $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ consists of all pairs $(x,y) \in [\![F_i]\!]_T(u) \times [\![F_i]\!]_T(v)$ such that $x =_v y$ holds and for which a simple path expression $F \sqsubseteq F_i$ with $x \in [\![F_i]\!]_T(u)$ and $y \in [\![F_i]\!]_T(v)$ exists.

As an example consider the XML tree in Figure 1 and suppose we replace the author nodes by firstauthor and secondauthor nodes. Suppose further we have a key $(\varepsilon, *.\text{article}, *.\text{lastname})$ with the *article* nodes as targets. The field $*$.lastname would then pick two lastname nodes for the first *article* node, and one lastname for the second *article* node. The value intersection of the two key node sets would contain the two lastname nodes for *Lindsay* as a pair, while the limited value intersection would be empty.

(Buneman, Davidson, Fan, Hara & Tan 2001, Buneman et al. 2003, Hartmann et al. 2007) study the axiomatisability and implication problem of the keys introduced in (Buneman et al. 2002). This time they restrict themselves to the path language PL consisting of all path expressions over the alphabet $\mathcal{L} \cup \{\_^*\}$. PL expressions are used for the key context C, the key selector Q and the key fields $F_1, \ldots, F_k$. In (Buneman, Davidson, Fan, Hara & Tan 2001) agreement of key node sets applies limited value intersection, while in (Buneman et al. 2003) it applies the original definition of value intersection.

All in all, (Buneman et al. 2002) define a key φ as an expression $(Q, (Q', Q_1, ..., Q_k))$ where Q, Q', Q_i are PL expressions such that $Q.Q'.Q_i$ is a valid PL expression for all $i = 1, \ldots, k$. Herein, Q is called the context path, Q' is called the target path, and $Q_1, ..., Q_k$ are called the key paths of φ. An XML tree T satisfies the key $(Q, (Q', Q_1, ..., Q_k))$ if and only if for any node $q \in [\![Q]\!]$: and for any nodes $q_1', q_2' \in q[\![Q']\!]$ such that there are nodes $x_i \in q_1'[\![Q_i]\!]$, $y_i \in q_2'[\![Q_i]\!]$ with $x_i =_v y_i$ for all $i = 1, \ldots, k$, then $q_1' = q_2'$ That is, $\forall q \in [\![Q]\!]$, $\forall q_1', q_2' \in q[\![Q']\!]$

$$\left(\bigwedge_{1 \leq i \leq k} q_1'[\![Q_i]\!] \cap_v q_2'[\![Q_i]\!] \neq \emptyset \right) \Rightarrow q_1' = q_2'$$

5.2 Keys defined by Arenas et al.

(Arenas et al. 2002, 2008) are motivated by the key notion of XML Schema, and study absolute and relative keys. The study focuses on strong keys and

Figure 2: XQuery Template for XML Key Validation

```
let $cp    := 'context path'
let $keys  := ('key1','key2','key3',...)
let $doc  := 'XML Document'
let $tp := 'target path' return
if (
    some $c in doc($doc)/saxon:evaluate($cp) satisfies (
        some $t1 in $c/saxon:evaluate($tp), $t2 in $c/saxon:evaluate($tp) satisfies (
            not($t1 is $t2) and (
            every $k in $keys satisfies (
                some $k1 in $t1/saxon:evaluate($k), $k2 in $t2/saxon:evaluate($k) satisfies (
                   fn:deep-equal($k1,$k2)
                )
            )
            )
         )
    )
) then 'key is violated'
else ('key is valid')
```

strong foreign keys. The context C and selector Q use path expressions $\_^*.\ell$ with $\ell \in E$, and key fields $F_1, ..., F_k$ with labels from A, where $k \geq 1$. In terms of node agreement, for target nodes they choose criterion (Ta), and for the key node, they apply criterion (Kd) for all $i = 1, ..., k$ (Arenas et al. 2002, 2008). Furthermore, key fields are limited to labels of attributes whose existence is guaranteed by a DTD or XML Schema. Meanwhile, the uniqueness of attributes is guaranteed by the XML standard (Bray et al. 2006). Therefore, $F_i(v)$ is a singleton set for every $i = 1, \ldots, k$, such that criteria (Tc, Td, Te) for the agreement of target nodes are automatically satisfied. In addition, (Arenas et al. 2008) further study absolute keys with a selector Q of the form $Q'.\ell$ where $\ell \in E$ and Q' denotes a regular expression over the alphabet $E \cup \{\_\}$. Their discussion is extended to absolute keys with path expressions as permitted by XML Schema (Arenas et al. 2002, Thompson et al. 2004). They show that the consistency problem is NP-hard for strong keys with $k = 1$ under non-recursive and no-star DTDs.

5.3 Keys defined by Yu and Jagadish

By studying data redundancies in XML, Yu and Jagadish (Yu & Jagadish 2008) focus on developing a partition based algorithm for discovering certain functional dependencies, which includes keys. Meanwhile, Yu and Jagadish define absolute keys for XML and their key notion has some similarity with the key from (Yu & Jagadish 2008). They have a different view on value equality, that is, they ignore the node order in the XML document; therefore, value equality in Yu and Jagadish's key definition is different to (Buneman et al. 2002) key definition, if the order of child elements is different. In addition, for the agreement of target nodes Yu and Jagadish choose criterion (Ta), and they choose criterion (Kc) for the agreement of key node sets for all $i = 1, \ldots, k$. Moreover, they use simple path expressions for selecting keys and key field expressions from the XPath fragment $XP(., /, ..)$, where key fields may use simple upward steps (Yu & Jagadish 2008).

6 Encoding XML Key Semantics in XQuery

In this section we present the first main contribution of our paper, which is a general XQuery template for the validation of XML documents against XML keys, as originally proposed by (Buneman et al. 2002). However, our template can also be applied to more expressive keys that feature expressions such as following, preceding, and ancestor axes, filters and disjunctions.

The core idea of this paper is to decide the validation problem for XML keys with the help of W3C-recommended XML standards, that is, XQuery (Chamberlin et al. 2010) and XML Schema 1.1 (Thompson et al. 2012). The benefits of this solution are potentially huge. As the XML keys, proposed by (Buneman et al. 2002), have been shown to naturally capture important application semantics, the means to actually validate XML documents effectively and efficiently would make it possible to apply these keys in everyday practice. This has important consequences as data exchange and integration become more effective with more consistent XML documents and as costs for data cleaning are reduced, for examples.

It is striking that the rather natural, yet complex, semantics of XML keys can almost directly be encoded with XQuery. More precisely, a key is assumed to be given in the format $(c, t, k_1, \ldots, k_n)$ where c denotes the context path of the key, t denotes the target path of the key, and $k_1, \ldots, k_n$ denote the key paths of the key. The key $(c, t, k_1, \ldots, k_n)$ is violated by the given XML document 'doc.xml' if and only if the following is true: there is some $\$c$ *in* doc('doc.xml')/$\$c$ and there are some $\$t_1$, $\$t_2$ in $\$c/\$t$ such that $\$t_1 \neq \$t_2$ and for every $\$k$ in $k_1, \ldots, k_n$ there exist some $\$v_1$ in $\$t_1/\$k$ and some $\$v_2$ in $\$t_2/\$k$ such that $\$v_1$ and $\$v_2$ are deep-equal. The above logic for validating the semantics of an XML key, as proposed by (Buneman et al. 2002), has been implemented in XQuery, as shown in Figure 2. For this purpose, the input of the validation problem is encoded in the form of four distinct variables. The context path of the key is stored in the variable $cp, the target path of the key in the variable $tp, and the key paths of the key are stored in the variable $keys. Finally, the file of the given XML document is stated in the variable $doc. The XQuery template is of the form `IF` condition holds `THEN` the key is vi-

olated `ELSE` the key is satisfied. The condition itself is a nested quantified expression that directly follows the semantics of the XML keys described above. In particular, it relies on the availability of such expressions as 'some', 'satisfies', 'not', and 'every', as well as the XQuery function 'deep-equal' that implements value-equality. The `SAXON` function 'evaluate' is necessary to create the XPath node selection queries at run-time. The normal XPath operator is not able to identify this string value and convert it to the actual node in the XML document. Therefore, we need to access the XML node dynamically. `Saxon:evaluate` is one of the extended functions in Saxon, which allows XPath expressions to be constructed and evaluated dynamically at run-time.

7 Performance Evaluation

In this section we present the second main contribution of this research. So far, we have demonstrated that semantics of expressive XML key notions can be validated effectively with XQuery. We will now measure how efficient the validation technique performs on differently sized real-world XML documents and with respect to XML keys that have different degrees of expressiveness. Firstly, we describe the feature of the real-world XML documents and the classes of XML keys we consider for the experiments. Secondly, we present and discuss the results of our performance analysis.

7.1 Input and Environment

Real-world XML documents. We applied our performance tests to various fragments of two real-world data sets, whose features we briefly describe now.

NASA is a set of astronomical data in XML format, converted from legacy flat-file format. It has been available to the public since 2001. It is not associated with a DTD. The size of this document is 23 Megabytes. When compared to the other data set, DBLP, that is used in this study, NASA has more depth and uses more attributes. The data set contains 476,646 element nodes, 56,317 attribute nodes, has a maximum depth of 8 and an average depth of 5.6.

The Digital Bibliography Library Project has brought forward the DBLP data set. It provides bibliographic information on major computer science journals and proceedings, and has been available since 2002. The DBLP data set is associated with a DTD and the size of the document is 127 Megabytes. The data set contains 3,332,130 element nodes, 404,276 attribute nodes, has a maximum depth of 6 and an average depth of 2.9.

For the experiments, we have extracted fragments from each NASA and DBLP that grow in size. There are 25 different fragments for each data source, with the next fragment including all previous ones plus new nodes. Each fragment represents an independent XML document.

Classes of XML Keys. We have evaluated the performance with respect to different classes of XML keys. The different classes result from the XPath axes they use:

(C1) Permits node labels and parent-child navigation,

(C2) Class (C1) extended by ancestor-descendant navigation in both context and target paths,

(C3) Class (C2) extended by label wildcard,

(C4) Class (C3) extended by following-sibling axis and filter expressions,

(C5) Class (C3) extended by preceding axis,

(C6) Class (C3) extended by union operator in field paths.

Experiments were conducted with the Professional Edition of Saxon 9.5 on the .NET platform on an Intel(R) Core(TM)i7 CPU M620 @2.66GHz Processor with 8GB of RAM and a 64-bit operating system.

7.2 Results

Table 1 shows the trends of time in seconds when applying our XQuery template for XML key validation to the nested fragments of the NASA and DBLP data sets, respectively. Each experiment was repeated 20 times, and the figures show minimum, average, and maximum times for each key and each data fragment.

Our performance analysis has covered keys that use different XPath axes and are applied to document fragments of various sizes. The performance analysis therefore allows us to draw a few conclusions regarding the efficiency of our proposed validation method. In Table 1 the general performance for the NASA XML document shows a linear growth. The main reasons for this trend are the size of the documents and the selectivity of the target nodes that need to be separated to satisfy the given key. Every given key on the NASA data set was either violated in every fragment or satisfied by every fragment. Based on our peformance analysis, we may conclude that our proposed implementation method works efficiently for medium-sized XML documents and complicated XML keys.

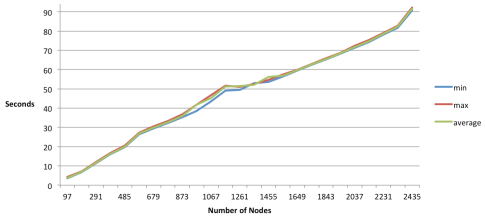
The performance of validating fragments of the DBLP data set against the given XML keys shows a different behavior. For many keys a first violation frequently appears in the middle of the fragments considered. Frequently, the validation time of subsequent fragments increases dramatically. Most likely, the reason for this dramatic increase is simply the number of target nodes that need to be separated. Ideally, once a violation occurs for a smaller fragment the violation of larger fragments would be found in a similar time. However, XPath first selects all target nodes (including new ones from larger fragments) before trying to separate them by the values on their key paths. We may thus conclude that once the number of target nodes that need to be separated becomes too large, validation times grow dramatically. Most likely this will occur in XML documents of large size.

8 Applying Assertions in XML Schema 1.1

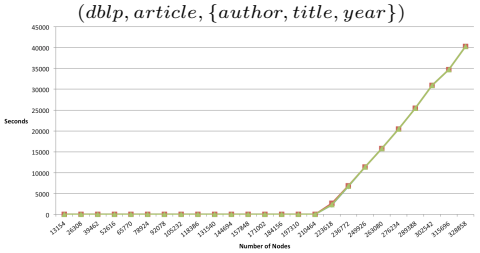
XML Schema 1.1 (Thompson et al. 2012) provides many more capabilities compared to XML Schema 1.0. Among those capabilities are assertions. By applying assertion, we are able to implement XML key validation in XML Schema 1.1. As we illustrate now, our XQuery implementation of the key by (Buneman et al. 2002) requires little change to be transformed into an XML Schema 1.1 (Thompson et al. 2012) assertion. Figure 3 contains a sample XML Schema definition and illustrates how an XML key can be expressed as an assertion to validate a given XML document.

We used oXygen XML Developer 15.0 to validate the sample XML document from Figure 3. The bottom of Figure 3 shows a screen-shots with the result of validating the document on the right against the XML Schema definition, inclusive of the XML key assertion,

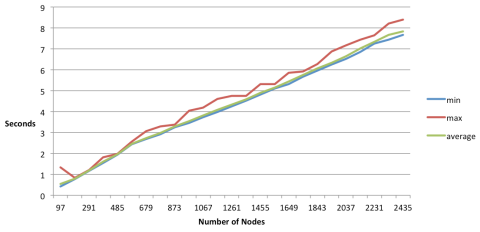
$(datasets, author, \{initial, lastName\})$



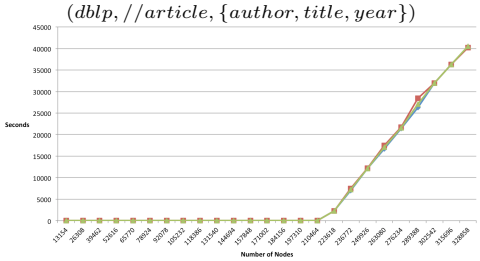
$(dblp, article, \{author, title, year\})$



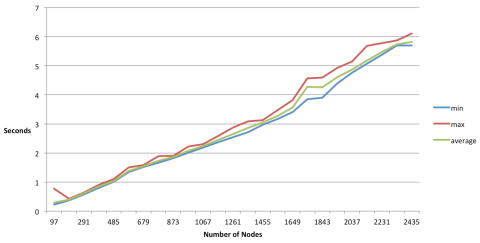
$(//dataset/history, //creator, \{lastName, affiliation\})$



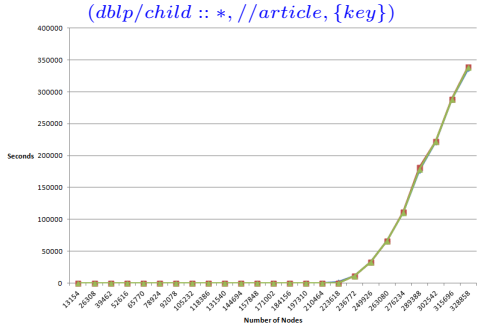
$(dblp, //article, \{author, title, year\})$



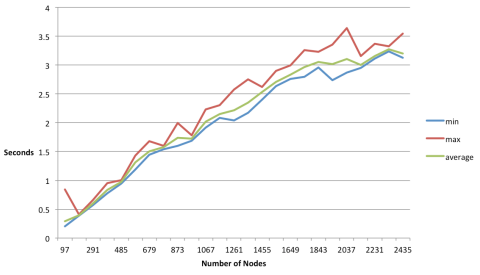
$(//datasets/reference/*, //*/other, \{title, author/*\})$



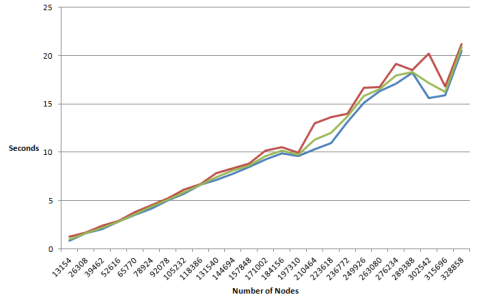
$(dblp/child::*, //article, \{key\})$



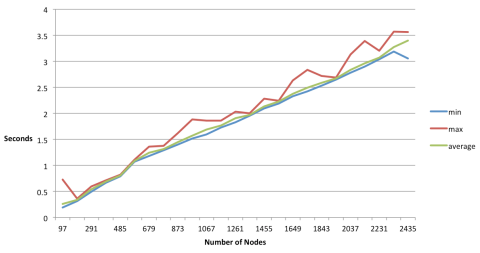
$(//dataset/following-sibling::*[5],$
$//author/following-sibling::*[1], \{initial, lastName\})$



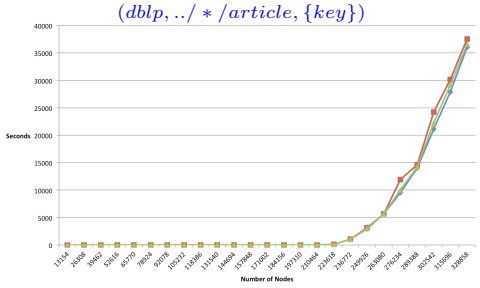
$(*, //www/following-sibling::*[8], \{title, url\})$



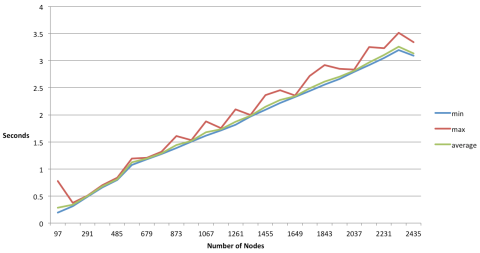
$(//dataset, //ingest, \{creator/preceding::*, date/preceding::*\})$



$(dblp, ../*/article, \{key\})$



$(datasets/child::*, //author, \{initial|lastName\})$



$(dblp, book, \{author, title|year\})$

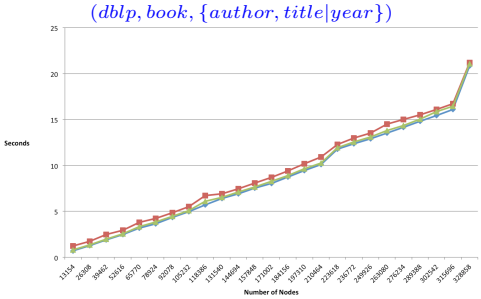
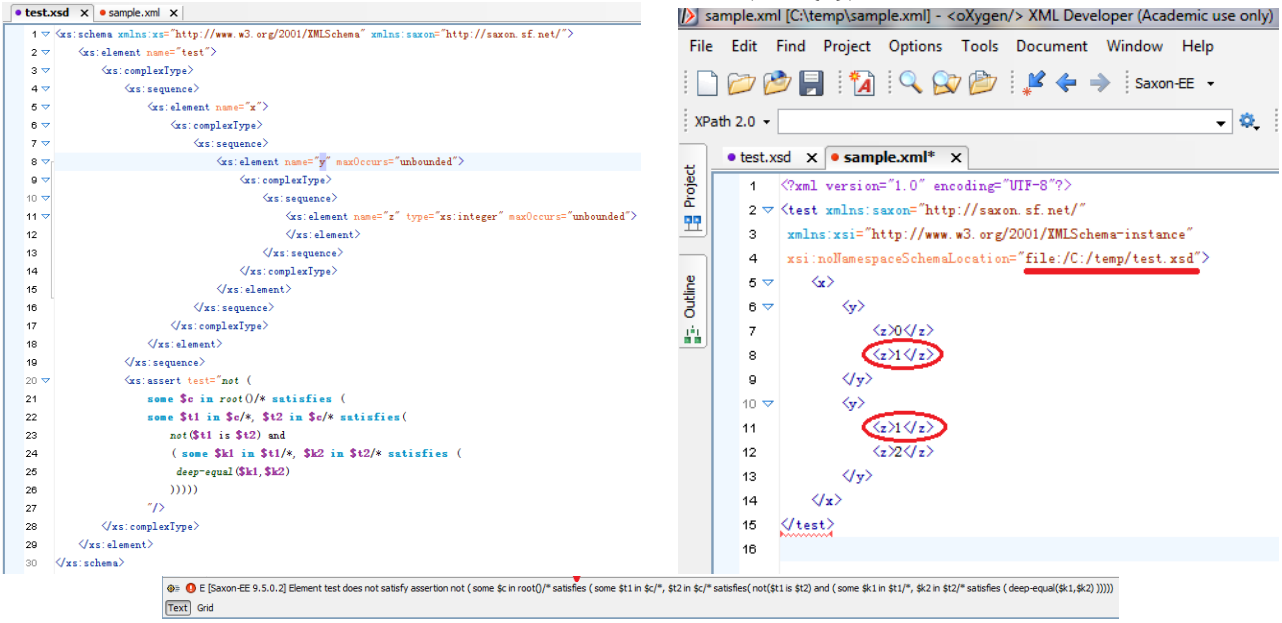


Table 1: Evaluating XQuery Validation of NASA and DBLP Data Sets Against XML Keys

Figure 3: XML Schema 1.1 with Assertion of XML Key $(x, y, \{z\})$ and Invalid XML Document



on the left. The document is invalid and the following error is reported 'Element test does not satisfy assertion not (some $(c) in root()/* satisfies (some $(t1) in $(c)/*, $(t2) in $(c)/* satisfies(not($(t1) is $(t2)) and (some $(k1) in $(t1)/*, $(k2) in $(t2)/* satisfies (deep-equal($(k1),$(k2)))))))'. The reason why the XML document is invalid is that there is an x-node that has two distinct y-children y_1 and y_2 such that y_1 has a z-child z_1 and y_2 has a z-child z_2 such that z_1 and z_2 are value-equal. On the other hand, if we change the value of the first z node in the second y node to 3, for example, the above XML document will be valid, because it satisfies the assertion.

9 Conclusion and Future Directions

The official W3C recommendation for XML keys requires the existence and uniqueness of nodes, which is a strong requirement for many applications and a source of intractability for associated reasoning tasks, which limits their applicability in practice. As a response, the academic community has proposed several different notions of XML keys which can capture expressive application semantics and have good computational properties. Our results have shown that keys of the most popular proposal (Buneman et al. 2002) can still be validated with off-the-shelf XML tools, including XQuery and XML Schema 1.1. This enables many organizations to express sophisticated application semantics within their XML documents, without the need to develop designated validation techniques. Therefore, better quality data and data-driven decision making can be obtained without use of additional resources. An analysis shows that our validation techniques perform rather well on medium-sized documents and sophisticated XML keys, but do not scale well on documents of large size. Further research is therefore required to find a sweet-spot for balancing well the simplicity and performance of validation.

References

Abiteboul, S., Buneman, P. & Suciu, D. (2000), *Data on the Web: From Relations to Semistructured Data and XML*, Morgan Kaufmann Publishers.

Abiteboul, S., Hull, R. & Vianu, V. (1995), *Foundations of Databases*, Addison-Wesley.

Abrão, M., Bouchou, B., Alves, M. H. F., Laurent, D. & Musicante, M. (2004), Incremental constraint checking for XML documents, *in* 'XSym 2004: Database and XML Technologies, Second International XML Database Symposium', Vol. 3186 of *LNCS*, Springer, pp. 112–127.

Apparao et al, V. (1998), 'Document object model (DOM) level 1 specification, W3C recommendation', `http://www.w3.org/TR/REC-DOM-Level-1/`.

Arenas, M., Fan, W. & Libkin, L. (2002), What's hard about XML schema constraints?, *in* A. Hameurlain, R. Cicchetti & R. Traunmüller, eds, 'DEXA 2002: Database and Expert Systems Applications, 13th International Conference', Vol. 2453 of *LNCS*, Springer, pp. 269–278.

Arenas, M., Fan, W. & Libkin, L. (2008), 'On the complexity of verifying consistency of XML specifications', *SIAM J. Comput.* **38**(3), 841–880.

Arenas, M. & Libkin, L. (2004), 'A normal form for XML documents', *Trans. Database Syst.* **29**(1), 195–232.

Arenas, M. & Libkin, L. (2005), 'An information-theoretic approach to normal forms for relational and XML data', *J. ACM* **52**(2), 246–283.

Benedikt, M., Fan, W. & Kuper, G. (2005), 'Structural properties of XPath fragments', *Theor. Comput. Sci.* **336**(1), 3–31.

Bouchou, B., Halfeld Ferrari Alves, M. & Musicante, M. (2003), Tree automata to verify XML key constraints, *in* 'WebDB 2003: International Workshop on Web and Databases', pp. 37–42.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E. & Yergeau, F. (2006), 'Extensible markup language (XML) 1.0 (fourth edition) W3C recommendation', `http://www.w3.org/TR/xml`.

Buneman, P., Davidson, S. B., Fan, W., Hara, C. S. & Tan, W. C. (2001), Reasoning about keys for XML, *in* 'DBPL'.

Buneman, P., Davidson, S., Fan, W., Hara, C. & Tan, W. (2002), 'Keys for XML', *Computer Networks* **39**(5), 473–487.

Buneman, P., Davidson, S., Fan, W., Hara, C. & Tan, W. (2003), 'Reasoning about keys for XML', *Inf. Syst.* **28**(8), 1037–1063.

Buneman, P., Fan, W., Siméon, J. & Weinstein, S. (2001), 'Constraints for semi-structured data and XML', *SIGMOD Record* **30**(1), 47–54.

Buneman, P., Fan, W. & Weinstein, S. (2000), 'Path constraints in semistructured databases', *J. Comput. Syst. Sci.* **61**(2), 146–193.

Chamberlin, D., Florescu, D., Robie, J., Simeon, J., Fernandez, M. & Boag, S. (2010), XQuery 1.0: An XML query language (second edition), W3C recommendation, W3C. http://www.w3.org/TR/2010/REC-xquery-20101214/.

Chen, Y., Davidson, S. & Zheng, Y. (2002), Xkvalidator: a constraint validator for XML, *in* 'CIKM 2002: Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management', ACM, pp. 446–452.

Clark, J. & DeRose, S. (1999), 'XML path language (XPath) version 1.0, W3C recommendation', `http://www.w3.org/TR/xpath`.

Deutsch, A. & Tannen, V. (2005), 'XML queries and constraints, containment and reformulation', *Theor. Comput. Sci.* **336**(1), 57–87.

Fan, W. (2005), XML constraints, *in* 'DEXA Workshops 2005: Proceedings of the 16th International Workshop on Database and Expert Systems Applications', IEEE Computer Society, pp. 805–809.

Fan, W. & Libkin, L. (2002), 'On XML integrity constraints in the presence of DTDs', *J. ACM* **49**(3), 368–406.

Fan, W. & Siméon, J. (2003), 'Integrity constraints for XML', *J. Comput. Syst. Sci.* **66**(1), 254–291.

Gottlob, G., Koch, C. & Pichler, R. (2005), 'Efficient algorithms for processing XPath queries', *ACM Trans. Database Syst.* **30**(2), 444–491.

Hartmann, S., Koehler, H., Link, S., Trinh, T. & Wang, J. (2007), On the notion of an XML key, *in* 'Proceedings of the 3rd International Workshop on Semantics in Data and Knowledge Bases (SDKB)', number 4925 *in* 'LNCS', Springer, pp. 103–112.

Hartmann, S. & Link, S. (2003), More functional dependencies for XML, *in* 'ADBIS', number 2798 *in* 'LNCS', Springer, pp. 355–369.

Hartmann, S. & Link, S. (2007), Unlocking keys for XML trees, *in* 'Proceedings of the 11th International Conference on Database Theory (ICDT)', number 4353 *in* 'LNCS', Springer, pp. 104–118.

Hartmann, S. & Link, S. (2009), 'Efficient reasoning about a robust XML key fragment', *ACM Trans. Database Syst.* **34**(2).

Hartmann, S. & Link, S. (2010), 'Numerical constraints on XML data', *Inf. Comput.* **208**(5), 521–544.

Karlinger, M., Vincent, M. W. & Schrefl, M. (2009), Keys in XML: Capturing identification and uniqueness, *in* 'Web Information Systems Engineering - WISE 2009, 10th International Conference, Poznan, Poland, October 5-7, 2009. Proceedings', Vol. 5802 of *Lecture Notes in Computer Science*, Springer, pp. 563–571.

Kay, M. (2009), XSL transformations (XSLT) version 2.0 (second edition), W3C proposed edited recommendation, W3C. http://www.w3.org/TR/2009/PER-xslt20-20090421/.

Liu, Y., Yang, D., Tang, S., Wang, T. & Gao, J. (2004), Extracting key value and checking structural constraints for validating XML key constraints, *in* 'WAIM 2004: 5th International Conference in Advances in Web-Age Information Management', Vol. 3129 of *LNCS*, Springer, pp. 399–408.

Liu, Y., Yang, D., Tang, S., Wang, T. & Gao, J. (2005), 'Validating key constraints over XML document using XPath and structure checking', *Future Generation Comp. Syst.* **21**(4), 583–595.

Miklau, G. & Suciu, D. (2004), 'Containment and equivalence for a fragment of XPath', *J. ACM* **51**(1), 2–45.

Neven, F. & Schwentick, T. (2006), 'On the complexity of XPath containment in the presence of disjunction, DTDs, and variables.', *Logical Methods in Computer Science* **2**(3:1).

Suciu, D. (2001), 'On database theory and XML', *SIGMOD Record* **30**(3), 39–45.

Thalheim, B. (1991), *Dependencies in Relational Databases*, Teubner.

Thalheim, B. (2000), *Entity-Relationship Modelling*, Springer.

Thompson, H., Beech, D., Maloney, M. & Mendelsohn, N. (2004), 'XML Schema Part 1: Structures Second Edition, W3C Recommendation', `http://www.w3.org/TR/xmlschema-1/`.

Thompson, H., Maloney, M., Sperberg-McQueen, M., Gao, S., Mendelsohn, N. & Beech, D. (2012), W3C xml schema definition language (XSD) 1.1 part 1: Structures, W3C recommendation, W3C. http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/.

Vianu, V. (2003), 'A web odyssey: from Codd to XML', *SIGMOD Record* **32**(2), 68–77.

Vincent, M., Liu, J. & Liu, C. (2004), 'Strong functional dependencies and their application to normal forms in XML', *Trans. Database Syst.* **29**(3), 445–462.

Vincent, M., Liu, J. & Mohania, M. (2007), 'On the equivalence between FDs in XML and FDs in relations', *Acta Inf.* **44**(3-4), 207–247.

Wood, P. (2003), Containment for XPath fragments under DTD constraints, *in* 'ICDT 2003: Proceedings of the 9th International Conference on Database Theory', number 2572 *in* 'LNCS', Springer, pp. 297–311.

Yu, C. & Jagadish, H. V. (2008), 'XML schema refinement through redundancy detection and normalization.', *VLDB J.* **17**(2), 203–223.