

On the Notion of an XML Key

Sven Hartmann^{1,*}, Henning Köhler², Sebastian Link³, Thu Trinh¹,
and Jing Wang⁴

¹ Clausthal University of Technology, Germany
`sven.hartmann@tu-clausthal.de`

² University of Queensland, Australia

³ Victoria University of Wellington, New Zealand

⁴ Massey University, New Zealand

Abstract. Ongoing efforts in academia and industry to advance the management of XML data have created an increasing interest in research on integrity constraints for XML. In particular keys have recently gained much attention. Keys help to discover and capture relevant semantics of XML data, and are crucial for developing better methods and tools for storing, querying and manipulating XML data. Various notions of keys have been proposed and investigated over the past few years. Due to the different ways of picking and comparing data items involved, these proposals give rise to constraint classes that differ in their expressive power and tractability of the associated decision problems. This paper provides an overview of XML key proposals that enjoy popularity in the research literature.

1 Introduction

XML, the eXtensible Markup Language [7] has become the standard for sharing and integrating data on the Web and elsewhere. There is wide consensus among researchers and practitioners that XML requires commensurate support by DBMS and data management tools, in particular to store, query and process XML data in its native format. The inherent syntactic flexibility and hierarchical structure of XML make it a challenge to precisely describe desirable properties of XML data and provide methods and facilities that can efficiently conclude, validate, or enforce such properties, cf. [15,16,17,26,28].

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 stored stays valid and does not deviate from reality. Integrity constraints enjoy a variety of applications ranging from schema design, query optimisation, efficient access and updating, privacy and integrity, data exchange and integration, to data cleaning, cf. [15].

Keys are without any doubt one of the most fundamental classes of integrity constraints. The importance of keys for XML has been recognized by industry

* Corresponding author.

and academia. They 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 [27]. While this has established an industry standard for specifying keys, Arenas et al. [2] have shown the computational intractability of the associated consistency problem, that is, the question whether there exists an XML document that conforms to a given DTD or XSD and satisfies the specified keys.

The most popular alternative proposal is due to Buneman et al [10,11] who define keys independently from any schema such as a DTD or XSD. Rather they based keys on the representation of XML data as trees. Keys uniquely identify nodes of interest in such a tree by some selected nodes, their associated complex values or their location.

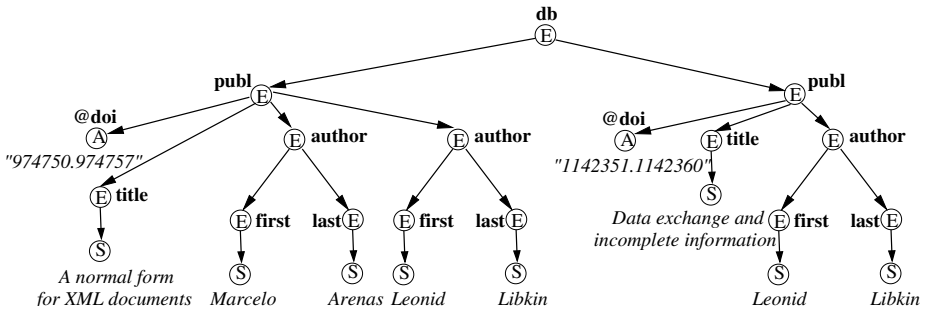


Fig. 1. An XML data tree

In Figure 1, an example of a reasonable key is that the *doi*-value identifies the *publ* node. That is, the *doi* subnodes of different *publ* 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 more than one publication. However, the *author* can indeed be identified by its *first* and *last* subnodes relatively to the *publ* node. That is, for each individual *publ* node, different *author* subnodes must differ on their *first* or *last* subnode value.

Clearly, the expressiveness of such keys depends on the means that are used for selecting nodes in a tree, and by the semantics of the associated node selection queries. The key notion proposed Buneman et al. is flexible in the choice of an appropriate query language. Unfortunately, increased expressive power often comes at the price of increased complexity of the associated decision problems. It is still a major challenge to find natural and useful classes of keys that can be managed efficiently [15,16,17,26,28].

The objective of this paper is to give a brief overview of definitions of keys that have been proposed in the research literature. Our focus is on the key notion of Buneman et al., but we also address alternative proposals [3,4,32].

2 Prerequisites

It is common to represent XML data by ordered, node-labelled trees. Such a representation is e.g. used in DOM [1], XPath [13], XQuery [6], XSL [22], and XML Schema [27]. Figure 1 shows an example of XML data represented as a tree in which nodes are annotated by their type: E for element, A for attribute, and S for text (PCDATA).

In the literature devoted to research on XML constraints, several variations of the tree model for XML data have been used. All of them simplify the XML standard [7] by concentrating on its major aspects. Here we follow the approach taken in [10,11]. We assume that there are three mutually disjoint non-empty sets \mathbf{E} , \mathbf{A} , and $\mathbf{S} = \{S\}$. In the sequel, \mathbf{E} will be used for element names, \mathbf{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*.

2.1 XML Trees and Paths

An *XML tree* is defined as a 6-tuple $T = (V, lab, ele, att, val, r)$. Herein V denotes the set of *nodes* of T , and lab is a mapping $V \rightarrow \mathcal{L}$ assigning a label to every node in V . The XML tree is said to be *finite* if V is finite. A node v in V is called an *element node* if $lab(v) \in \mathbf{E}$, an *attribute node* if $lab(v) \in \mathbf{A}$, and a *text node* if $lab(v) \in \mathbf{S}$. Further, 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 . Further, val is a partial mapping assigning a string value to each attribute and text node: for any node v in V , if v is an attribute or text node then $val(v)$ is a string, and $val(v)$ is undefined otherwise. Finally, 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 \mathbf{M} . The unique sequence of zero symbols is called the *empty path expression*, denoted by ε , and a dot (\cdot) is used to denote the concatenation of path expressions. The set of all path expressions over \mathbf{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, \dots, v_m where (v_{i-1}, v_i) is an edge for $i = 1, \dots, m$. 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 $lab(v_1) \cdot \dots \cdot lab(v_m)$, 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 .

2.2 Value Equality

Next we 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 text or attribute nodes, or their children are pairwise value equal if they are element nodes. 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. That is, two nodes u and v are value equal when the following conditions are satisfied:

- (a) $lab(u) = lab(v)$,
- (b) if u, v are attribute or text nodes, then $val(u) = val(v)$,
- (c) if u, v are element nodes, then (i) if $att(u) = \{a_1, \dots, a_m\}$, then $att(v) = \{a'_1, \dots, a'_m\}$ and there is a permutation π 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 $i = 1, \dots, k$.

For example, the second and third *author* node (according to document order) in Figure 1 are value equal. Notice 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.

Consider two subsets U and W of V . We call U and W *value equal* if there exists a bijection $\beta : U \rightarrow W$ 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 .

2.3 Node Selection Queries

A *node selection query* Q defines a mapping $\llbracket Q \rrbracket_T : V \rightarrow 2^V$ 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, absoluton and the identity as follows, cf. [13]:

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

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 $\llbracket Q \rrbracket_T(v)$ is a subset of $\llbracket Q' \rrbracket_T(v)$. 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 [13] 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 [5,14,18,23,24,25,31].

3 Keys for XML

In this section we attempt to give a definition of keys for XML data that is sufficiently general to cover most of the existing proposals for defining such constraints. We should note that this definition might not always provide the most convenient way of expressing the constraint at hand nor reveal the most efficient way for tackling the associated decision problems.

An *XML key* is a pair $\sigma = (C, Q, \mathcal{F})$ where C and Q are node selection queries, and $\mathcal{F} = \{F_1, \dots, F_k\}$ is a finite set of node selection queries. Adapting the terminology of [27], we call C the *context*, Q the *selector* and F_1, \dots, F_k the *fields* of the key σ . Given an XML tree T , $\llbracket C \rrbracket_T(r)$ is called the *context node set*. For any context node u , $\llbracket Q \rrbracket_T(u)$ is called the *target node set*, and for any target node v and every $i = 1, \dots, k$ we call $\llbracket F_i \rrbracket_T(v)$ a *key node set*.

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 $i = 1, \dots, k$ their key node sets $\llbracket F_i \rrbracket_T(u)$ and $\llbracket F_i \rrbracket_T(v)$ *agree* then the nodes u and v themselves *agree*.

3.1 Agreement of Nodes

In the literature different authors have suggested different approaches for defining *agreement* of target nodes and of key node sets. To continue with we will summarize the most popular proposals. For the *agreement* of two key node sets the following criteria are potentially of interest:

- (Ka) $\llbracket F_i \rrbracket_T(u)$ and $\llbracket F_i \rrbracket_T(v)$ are equal,
- (Kb) $\llbracket F_i \rrbracket_T(u)$ and $\llbracket F_i \rrbracket_T(v)$ have non-empty intersection,
- (Kc) $\llbracket F_i \rrbracket_T(u)$ and $\llbracket F_i \rrbracket_T(v)$ are value equal,
- (Kd) $\llbracket F_i \rrbracket_T(u)$ and $\llbracket F_i \rrbracket_T(v)$ have non-empty value intersection.

For the *agreement* of two target nodes u and v the following criteria are potentially of interest:

- (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 $i = 1, \dots, k$:

- (Tc) both $\llbracket F_i \rrbracket_T(u)$ and $\llbracket F_i \rrbracket_T(v)$ are non-empty,
- (Td) both $\llbracket F_i \rrbracket_T(u)$ and $\llbracket F_i \rrbracket_T(v)$ contain at most one node,
- (Te) both $\llbracket F_i \rrbracket_T(u)$ and $\llbracket F_i \rrbracket_T(v)$ contain only attribute or text nodes.

3.2 Strong Keys Defined in XML Schema

As an example we consider keys as defined in XML Schema [27]. For such a key σ to hold it is necessary that for each target node v each of the key fields F_i with $i = 1, \dots, k$ selects exactly one key node. This prerequisite calls for uniqueness and existence. Buneman et. al [9] call a key with such a prerequisite *strong*.

XML Schema [27] 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 $\llbracket F_i \rrbracket_T(v)$ 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*. For such a constraint σ to hold it is necessary that for each target node v each of the key fields F_i with $i = 1, \dots, k$ selects no more than one key node. Again criterion (Kd) is used for the agreement of key node sets, and criteria (Ta,Td,Te) for the agreement of target nodes.

3.3 Absolute and Relative Keys

A key of the form $(\varepsilon, Q, \mathcal{F})$ is called *absolute*. That is, in case of an absolute key the root node is the single context node. To emphasize that in general the key context C may also be chosen differently, one speaks of *relative* keys. It should be noted that every relative key may be rewritten as an absolute key by transforming the key context C into an additional key field with criterion (Kb) 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.

4 Popular Existing Proposals for XML Keys

4.1 Keys Defined by Buneman et al.

In [8,10] Buneman et al. introduce and study absolute and relative keys that do not have the uniqueness and existence prerequisite of the strong keys defined in

XML Schema [27]. This proposal is mainly motivated by the observation that strong keys are not always finitely satisfiable. That is, there are keys for which there is no a finite XML tree that satisfy them. To overcome this situation, Buneman et al. 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 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. Herein, $_$ and $_*$ are symbols not in \mathcal{L} that serve as the *single symbol wildcard* and the *variable length wildcard*. The semantics of path expressions from PE is defined by putting:

$$\begin{aligned} \llbracket \ell \rrbracket_T(v) &:= \{w \mid (v, w) \in E_T, \text{lab}_T(w) = \ell\} \\ \llbracket _ \rrbracket_T(v) &:= \{w \mid (v, w) \in E_T\} \\ \llbracket _ * \rrbracket_T(v) &:= \{w \mid (v, w) \in E_T^*\} \end{aligned}$$

The semantics of the concatenation operator and of ε is defined as for general node selection queries. When comparing PE and XPath [13], one observes the equivalences $\varepsilon \equiv \cdot$ and $_ \equiv *$ and $_ * \equiv _ / _$ and $Q_1 \cdot Q_2 \equiv Q_1 / Q_2$ and $Q_1 // Q_2 \equiv Q_1 \cdot _ * \cdot Q_2$ showing that PE corresponds to the XPath fragment $XP(\cdot, /, *, //)$.

In [8,10] PE expressions are used for the context C and the key selector Q , while simple path expressions from are used for the key fields F_1, \dots, F_k . At the end of [10] 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* $\llbracket F_i \rrbracket_T(u) \cap_{lv} \llbracket F_i \rrbracket_T(v)$ of key node sets $\llbracket F_i \rrbracket_T(u)$ and $\llbracket F_i \rrbracket_T(v)$ consists of all pairs $(x, y) \in \llbracket F_i \rrbracket_T(u) \times \llbracket F_i \rrbracket_T(v)$ such that $x =_v y$ holds and for which a simple path expression $F \sqsubseteq F_i$ with $x \in \llbracket F_i \rrbracket_T(u)$ and $y \in \llbracket F_i \rrbracket_T(v)$ 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 $(\varepsilon, _ * \cdot \text{publ}, _ * \cdot \text{last})$ with the *publ* nodes as targets. The field $_ * \cdot \text{last}$ would then pick two *last* nodes for the first *publ* node, and one *last* for the second *publ* node. The value intersection of the two key node sets would contain the two *last* nodes for Libkin as a pair, while the limited value intersection would be empty.

In [9,11] Buneman et al. study the axiomatisability and implication problem of the keys introduced in [8,10]. 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, \dots, F_k . In [9] agreement of key node sets is based on limited value intersection, while in [11] it is based on the original definition of value intersection.

Several authors have continued to investigate relevant properties for this class of keys. Without claiming to be complete we will mention a few. In [12] Chen et al. study efficient ways for validating keys against an XML tree. Their procedure can be used for checking the semantic correctness of an XML tree and for checking incremental updates made to an XML tree. Its asymptotic performance is linear in the size of the input tree and the input keys.

In [19] Grahne and Zhu develop efficient methods for mining keys that are satisfied by an XML tree. The keys they search for use path expressions over the alphabet $\mathcal{L} \cup \{-\}$. Their algorithm also looks for keys that hold only in certain parts of the tree. These approximate keys may serve as candidate keys for views.

In [29] Vincent et al. use absolute keys with simple path expressions as selectors and fields F_1, \dots, F_k and with $k \geq 1$. In particular, they emphasize that such a key corresponds to a strong functional dependency as defined and studied in [29]. Note that Buneman et al. [11] give an example of a key with $k = 0$ and emphasize that such a key may cause inconsistency in the presence of a DTD.

In [21] Hartmann and Link give an axiomatisation for the class of keys where context C and selector Q are *PL* expressions, and the fields F_i are simple path expressions with $k \geq 1$. The completeness proof is based on a characterisation of key implication in terms of reachability of nodes in a suitable digraph constructed from a key to reason about. Utilising the efficient evaluation of Core XPath queries [18] this gives rise to a decision procedure for the implication of this key class that is quadratic in the size of the input keys.

More recently, Wang [30] has investigated the axiomatisability and implication problem for keys where the path expressions may include the single symbol wildcard $-$. Her results exploit a correspondence between XML keys and tree patterns that have been used by Miklau and Suciu [23] to study containment problem for certain XPath fragments.

In [20] Hartmann and Link extend XML keys to numerical constraints for XML. While keys are intended to uniquely determine nodes in an XML tree, numerical constraints only do this up to a certain number of nodes. For this number one may specify bounds. The implication problem is shown to be intractable for some classes of numerical constraints. For the class of *numerical keys* with arbitrary integers as upper bounds and with *PL* expressions as context and selector, simple path expressions as fields, and $k \geq 1$, however, implication can be decided in quadratic time using shortest path methods.

4.2 Keys Defined by Arenas et al.

Motivated by the key notion of XML Schema, Arenas et al. [2,3,4] study absolute and relative keys. The focus is on the investigation of consistency problems for strong keys (and strong foreign keys). As context C and selector Q they use path expressions of the form $-^*.\ell$ with $\ell \in \mathbf{E}$, and as key fields F_1, \dots, F_k labels from \mathbf{A} , where $k \geq 1$. For the agreement of target nodes they choose criterion (Ta), and for the agreement of key node sets criterion (Kd) for all $i = 1, \dots, k$. It should be noted that key fields are limited to the labels of attributes whose existence is guaranteed by a DTD (or XSD). At the same time the uniqueness of attributes is guaranteed by the XML standard [7]. Thus, $F_i(v)$ is a singleton set for every $i = 1, \dots, k$, such that criteria (Tc, Td, Te) for the agreement of target nodes are automatically satisfied.

In [3,4] Arenas et al. further study absolute keys with a selector Q of the form $Q'.\ell$ where $\ell \in \mathbf{E}$ and Q' denotes a regular expression over the alphabet $E \cup \{-\}$. In [2] the discussion is extended to absolute keys with path expressions as permitted

by XML Schema [27]. In particular, Arenas et al. show that the consistency problem is NP-hard already for strong keys with $k = 1$ in the presence of a non-recursive and no-star DTD.

4.3 Keys Defined by Yu and Jagadish

In [32] Yu and Jagadish study data redundancies in XML that are caused by functional dependencies. The focus is on the development of an efficient partition-based algorithm for discovering certain functional dependencies, including keys.

For that they also give a definition of absolute keys for XML that shares some similarity with the notion of Buneman et al. First of all, [32] does not take document order into account. This leads to a different notion of value equality: Two nodes that are value equal in [32] might not be value equal by the original definition above if the order of their element children differ. For the agreement of target nodes Yu and Jagadish choose criterion (Ta), and criterion (Kc) for the agreement of key node sets for all $i = 1, \dots, k$. For the key selector they use simple path expressions, and for the key fields expressions from the XPath fragment $XP(., /, ..)$. That is, key fields may involve simple upward steps.

Acknowledgement

We like to thank Bernhard Thalheim for his constant encouragement and support during the preparation of this paper.

References

1. Apparao, V., et al.: Document object model (DOM) level 1 specification, W3C recommendation (October 1998), <http://www.w3.org/TR/REC-DOM-Level-1/>
2. Arenas, M., Fan, W., Libkin, L.: What's hard about XML schema constraints? In: Hameurlain, A., Cicchetti, R., Traunmüller, R. (eds.) DEXA 2002. LNCS, vol. 2453, pp. 269–278. Springer, Heidelberg (2002)
3. Arenas, M., Fan, W., Libkin, L.: Consistency of XML specifications. In: Akiyama, J., Baskoro, E.T., Kano, M. (eds.) IJCCGGT 2003. LNCS, vol. 3330, pp. 15–41. Springer, Heidelberg (2005)
4. Arenas, M., Fan, W., Libkin, L.: On the complexity of verifying consistency of XML specifications. *SIAM J. Comput.* 38(3), 841–880 (2008)
5. Benedikt, M., Fan, W., Kuper, G.M.: Structural properties of XPath fragments. *Theor. Comput. Sci.* 336(1), 3–31 (2005)
6. Boag, S., Chamberlin, D., Fernández, M., Florescu, D., Robie, J., Siméon, J.: XQuery 1.0: An XML query language, W3C Proposed Recommendation (November 2006), <http://www.w3.org/TR/xquery/>
7. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible markup language (XML) 1.0 (3rd edn.), W3C Recommendation (February 2004), <http://www.w3.org/TR/2004/REC-xml-20040204/>
8. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Keys for XML. In: WWW, vol. 10 (2001)

9. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Reasoning about keys for XML. In: DBPL, pp. 133–148 (2001)
10. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Keys for XML. *Computer Networks* 39(5), 473–487 (2002)
11. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Reasoning about keys for XML. *Information Systems* 28(8), 1037–1063 (2003)
12. Chen, Y., Davidson, S.B., Zheng, Y.: XKvalidator: a constraint validator for XML. In: CIKM 2002 (2002)
13. Clark, J., DeRose, S.: XML path language (XPath) version 1.0, W3C Recommendation (November 1999), <http://www.w3.org/TR/xpath>
14. Deutsch, A., Tannen, V.: Containment and integrity constraints for XPath. In: KRDB (2001)
15. Fan, W.: XML constraints. In: DEXA Workshops, pp. 805–809 (2005)
16. Fan, W., Libkin, L.: On XML integrity constraints in the presence of DTDs. *J. ACM* 49(3), 368–406 (2002)
17. Fan, W., Siméon, J.: Integrity constraints for XML. *J. Comput. Syst. Sci.* 66(1), 254–291 (2003)
18. Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. *ACM Trans. Database Syst.* 30(2), 444–491 (2005)
19. Grahne, G., Zhu, J.: Discovering approximate keys in XML data. In: CIKM 2002: Proceedings of the eleventh international conference on Information and knowledge management, pp. 453–460. ACM, New York (2002)
20. Hartmann, S., Link, S.: Numerical constraints for XML. In: Leivant, D., de Queiroz, R. (eds.) WoLLIC 2007. LNCS, vol. 4576, pp. 203–217. Springer, Heidelberg (2007)
21. Hartmann, S., Link, S.: Unlocking keys for XML trees. In: Schwentick, T., Suciú, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 104–118. Springer, Heidelberg (2006)
22. Kay, M.: XSL transformations (XSLT) version 2.0 W3C Candidate Recommendation (November 2005), <http://www.w3.org/TR/xslt20/>
23. Miklau, G., Suciú, D.: Containment and equivalence for a fragment of XPath. *J. ACM* 51(1), 2–45 (2004)
24. Neven, F., Schwentick, T.: XPath containment in the presence of disjunction, DTDs, and variables. *Logical Methods in Computer Science* 2(3) (2006)
25. Schwentick, T.: XPath query containment. *SIGMOD Record* 33(1), 101–109 (2004)
26. Suciú, D.: On database theory and XML. *SIGMOD Record* 30(3), 39–45 (2001)
27. Thompson, H., Beech, D., Maloney, M., and Mendelsohn, N.: XML Schema part 1: Structures second edition, W3C Recommendation, (October 2004), <http://www.w3.org/TR/xmlschema-1/>
28. Vianu, V.: A web odyssey. *SIGMOD Record* 32, 68–77 (2003)
29. Vincent, M., Liu, J., Liu, C.: Strong functional dependencies and their application to normal forms in XML. *ACM ToDS* 29(3), 445–462 (2004)
30. Wang, J.: Using tree patterns for flexible handling of XML keys. Master’s thesis, Massey University (2008)
31. Wood, P.T.: Containment for XPath fragments under DTD constraints. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572, pp. 297–311. Springer, Heidelberg (2002)
32. Yu, C., Jagadish, H.V.: Efficient discovery of XML data redundancies. In: VLDB, pp. 103–114 (2006)