Keys in XML: Capturing Identification and Uniqueness

Michael Karlinger[1], Millist Vincent[2], and Michael Schrefl[1]

[1] Johannes Kepler University, Linz, Austria
{karlinger,schrefl}@dke.uni-linz.ac.at
[2] University of South Australia, Adelaide, Australia
millist.vincent@unisa.edu.au

Abstract. In this article a new type of key constraint in XML, called an *XKey*, is proposed. The motivation for an XKey is based on the observation that existing approaches do not always capture the fundamental properties of a key, namely identification and uniqueness, and it is shown that an XKey always has these properties. It is also shown that an XKey has the desirable property of extending the notion of a relational key.

Keywords: XML, Integrity Constraints, Keys.

1 Introduction

Integrity constraints are one of the oldest and most important topics in database research, and they find application in a variety of areas such as database design, data translation and data storage [1]. With the adoption of XML [2] as the industry standard for data interchange over the internet, and the increasing usage of XML as a format for the permanent storage of data in database systems [3], the study of integrity constraints in XML has increased in importance in recent years (cf. [4] for a recent survey of the topic).

While many different types of integrity constraints have been proposed and studied, the most important type of integrity constraint is probably a key constraint, irrespective of the data model used, since it is the fundamental means by which entities in a database can be identified. A very simple type of a key constraint for XML is offered by DTDs [2] in the form of an `Id` constraint. More sophisticated key frameworks have recently been proposed in [5], and in the `key` and the `uniqueness` constraints of XML Schema [2]. In these approaches, a key is syntactically defined by a statement of the form $(T, (P_1, \ldots, P_n))$, where T is a path referred to as the *selector* and $P_1, \ldots, P_n$ are paths called *fields*, and the semantics of a key then requires that T nodes in the XML tree (document) are identified by the combination of $P_1, \ldots, P_n$ nodes.

While these approaches have made an important contribution to the specification and study of XML keys, they have several important limitations that we now highlight in the following example. Suppose that we wish to store information about customers of a phone company. Two sample XML documents are shown in Fig. 1. Suppose also that the application specifies the constraint that customers are identified by the `code` and `number` of their phones,

G. Vossen, D.D.E. Long, and J.X. Yu (Eds.): WISE 2009, LNCS 5802, pp. 563–571, 2009.

(a)

```
<Customers>
 <Customer name="Jones">
  <Phone code="0660" number="44444"/>
 </Customer>
 <Customer name="Smith">
  <Phone code="0660" number="11111"/>
  <Phone code="0990" number="44444"/>
 </Customer>
</Customers>
```

(b)

```
<Customers>
 <Customer name="Miller">
  <Phone code="0880" number="33333"/>
  <Phone code="0880" number="33333"/>
 </Customer>
</Customers>
```

Fig. 1. Example XML Documents Representing Customers and their Phones

i.e. the key is $\kappa = (\texttt{Customers.Customer}, (\texttt{Phone.code}, \texttt{Phone.number}))$, where `Customers.Customer` is the selector and `Phone.code` and `Phone.number` are the fields.

In reference to the approach of [5], its first limitation is that while it is intended to capture the *identification property* of a key, it does not always do this, as we now explain. From an intuitive view point, the XML document in Fig. 1a satisfies κ since the combination of `code` plus `number` of each phone is unique for the two customers. However, according to the semantics of [5], when there are multiple fields and there is more than one field node per field then all possible combinations of field nodes are required to identify the selector node. Hence κ is violated in Fig. 1a according to the semantics in [5], since both `customer` nodes have the field node combination of `0660` and `44444`.

The second limitation of the approach in [5] is that it does not capture, nor was intended to capture, the *uniqueness property* of a key, which we now illustrate. In the XML document in Fig. 1b, key κ is satisfied according to the semantics in [5], which is however not desirable since the phone `0880/33333` is stored twice. This is a disadvantage, since having a key that is not unique results in redundancy and update problems similar to what occurs in relational databases. For example, if the `code` or `number` of one of the phones of `Miller` is modified but not the other, then the document becomes inconsistent.

XML Schema provides two types of identification constraints, a `key` constraint and a `unique` constraint, with the intention that a `key` constraint correspond to a primary key and a `unique` constraint correspond to a candidate key. The two constraints are both specified using the same syntax (given earlier), but have slightly different semantics. While a `key` constraint specifies that there must be at least one field node per field, a `unique` constraint allows the field nodes to be empty. Both constraints however require that, for each selector node and each field, there is at most one field node. So, for example, in Fig. 1a this would require that a customer can only have at most one phone, and so κ would be violated even though the combination of `code` plus `number` of each phone is unique and effectively identifies the customer it belongs to. We regard this approach as being too strict and not essential for capturing the identification

(a)

cno	code	number
C1	0660	44444
C2	0660	11111
C1	0770	22222
C2	0990	44444

(b)

cno	{code	number}
C1	0660	44444
	0770	22222
C2	0660	11111
	0990	44444

(c)

```
<Customers>
  <Customer cno="C1">
    <Phone code="0660" number="44444"/>
    <Phone code="0770" number="22222"/>
  </Customer>
  <Customer cno="C2">
    <Phone code="0660" number="11111"/>
    <Phone code="0990" number="44444"/>
  </Customer>
</Customer>
```

Fig. 2. Relation `Phones` **(a)** flat, **(b)** nested, and **(c)** mapped to XML

and uniqueness property of a key, and so we will later define a key constraint differently so that κ is satisfied in Fig. 1a (but not in Fig. 1b).

The second limitation of the work in [5], and the XML Schema constraints is that they do not extend the semantics of a relational key. To illustrate this, consider the following example, closely related to the one given earlier.

The relation shown in Fig. 2a stores details of phones and the customers that they belong to. The key for relation `Phones` is (`code`, `number`), and so `code` plus `number` identifies the number of a customer (`cno`), but `cno` does not identify `code` and `number` since a customer may have more than one phone.

We now map relation `Phones` to an XML document by first nesting on {`code`, `number`}, and then directly to an XML document as shown in Fig. 2. This mapping is an instance of a general method of mapping a relation to an XML document, which was originally presented in [6]. We also name the XML document as `Customers`, rather than `Phones`, to reflect the fact that the details of phones are now nested under `cno` as a result of the mapping. As a result of the nesting, `cno` is now unique in `Customers` and so, as one would expect, the key (`Customers.Customer`, (`cno`)) holds in `Customers`. However, since (`code`, `number`) is a key in `Phones` one would also expect that (`code`, `number`) identifies customers in `Customers`, and hence that the key (`Customers.Customer`, (`Phone.code,Phone.number`)) would hold. However, this is not the case under either the semantics of [5] or XML Schema, for the same reasons that the XML document in Fig. 1a violates κ.

Having an XML key constraint that extends the semantics of a relational key is important in several areas. Firstly, in the area of XML publishing [7], where a source relational database has to be mapped to a single predefined XML schema, knowing how relational integrity constraints map to XML integrity constraints allows the XML document to preserve more of the original semantics. This argument also applies to 'data-centric' XML [8], where XML databases (not necessarily with predefined schemas) are generated from relational databases.

The first contribution of our paper is to propose a new key constraint for XML, called an *XKey*. An XKey specifies that field nodes related by a semantic

property called *closest* are required to be unique, thus obviously capturing the uniqueness property and eliminating the redundancy problems discussed earlier. However, we also show that a consequence of the uniqueness property of an XKey is that selector nodes are identified by field nodes, and so an XKey simultaneously captures both the uniqueness and identification properties of a key.

Our second contribution is to show that an XKey extends the semantics of a relational key, thus eliminating the third limitation of the existing approaches discussed earlier. We do this by showing that in the special case where the XML tree is derived from a flat relation by first mapping it to a nested relation by an arbitrary sequence of nest operations, then the XML tree satisfies an XKey if the flat relation satisfies the corresponding relational key.

The rest of the paper is organized as follows. Section 2 contains preliminary definitions, and Sect. 3 contains the definition of our XKey. Finally, Sect. 4 gives an overview over related work.

2 XML Trees, Paths and Reachable Nodes

In this section we present some preliminary definitions. First, following the model adopted by XPath and DOM [2], we model an XML document as a tree as follows. We assume countably infinite, disjoint sets $\mathbf{E}$ and $\mathbf{A}$ of element and attribute labels respectively, and the symbol $\mathcal{S}$ indicating text. Thereby, the set of labels that can occur in the XML tree, $\mathbf{L}$, is defined by $\mathbf{L} = \mathbf{E} \cup \mathbf{A} \cup \{\mathcal{S}\}$.

Definition 1. *An* XML tree $\mathbb{T}$ *is defined by* $\mathbb{T} = (\mathbf{L}, \mathbf{V}, E, \mathrm{lab}, \mathrm{val}, v_\rho)$*, where*

- $\mathbf{V}$ *is a finite, non-empty set of nodes;*
- *the function* $\mathrm{lab} : \mathbf{V} \to \mathbf{L}$ *assigns a label to every node in* $\mathbf{V}$*. A node* v *is called an* element node *if* $\mathrm{lab}(v) \in \mathbf{E}$*, an* attribute node *if* $\mathrm{lab}(v) \in \mathbf{A}$*, and a* text node *if* $\mathrm{lab}(v) = \mathcal{S}$*;*
- $v_\rho \in \mathbf{V}$ *is a distinguished element node, called the* root node*, and* $\mathrm{lab}(v_\rho) = \rho$*;*
- *the parent-child relation* $E \subset V \times V$ *defines the directed edges connecting the nodes in* $\mathbf{V}$ *and is required to form a tree structure rooted at node* v_ρ*. Thereby, for every edge* $(v, \bar{v}) \in E$*,*
  1. v *is an* element node *and is said to be the* parent *of* $\bar{v}$*. Conversely,* $\bar{v}$ *is said to be a* child *of* v*;*
  2. *if* $\bar{v}$ *is an attribute node, then there does not exist a node* $\tilde{v} \in \mathbf{V}$ *and an edge* $(v, \tilde{v}) \in E$ *such that* $\mathrm{lab}(\tilde{v}) = \mathrm{lab}(\bar{v})$ *and* $\tilde{v} \neq \bar{v}$*;*
- *the partial function* $\mathrm{val} : \mathbf{V} \to$ *string assigns a string value to every attribute and text node in* $\mathbf{V}$*.*

We also denote the parent of node v in a tree by $\mathrm{parent}(v)$, and the set of ancestor nodes of v by $\mathrm{ancestor}(v)$. An example of an XML tree is presented in Fig. 3, which is the tree representation of the XML document in Fig. 2, where $\mathbf{E} = \{\rho, \texttt{Customer}, \texttt{Phone}\}$ and $\mathbf{A} = \{\texttt{cno}, \texttt{code}, \texttt{number}\}$.

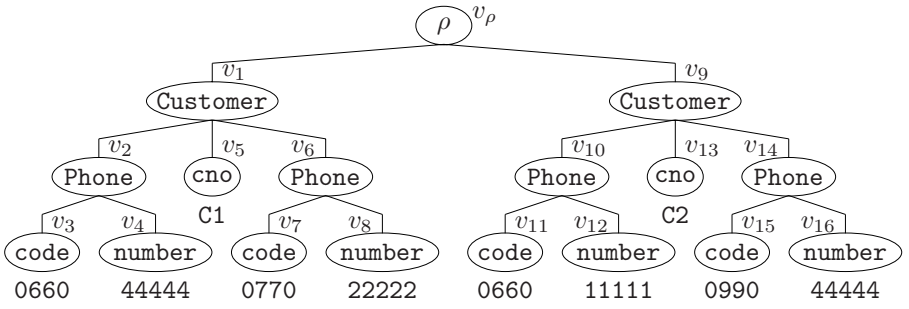

Fig. 3. Tree Representation of the XML Document in Fig. 1c

The notion of a path, which we now present together with some frequently required operators on paths, is central to all work on XML integrity constraints.

Definition 2. *A* path $P = l_1.\cdots.l_n$ *is a non-empty sequence of labels (possibly with duplicates) from* $\mathbf{L}$*. Given paths* $P = l_1.\cdots.l_n$ *and* $\bar{P} = \bar{l}_1.\cdots.\bar{l}_m$ *we define*

- P *to be a* legal *path, if* $l_1 = \rho$ *and* $l_i \in \mathbf{E}\ \forall i \in [1, n-1]$[1].
- P *to be a* prefix *of* $\bar{P}$*, denoted by* $P \subseteq \bar{P}$*, if* $n \leq m$ *and* $l_i = \bar{l}_i\ \forall i \in [1, n]$.
- *the* concatenation *of* P *and* $\bar{P}$*, denoted by* $P.\bar{P}$*, to be* $l_1.\cdots.l_n.\bar{l}_1.\cdots.\bar{l}_m$.
- *the* intersection *of* P *and* $\bar{P}$ *if both are legal paths, denoted by* $P \cap \bar{P}$*, to be the longest path that is a prefix of both* P *and* $\bar{P}$.
- $\text{parent}(P) = l_1.\cdots.l_{n-1}$*, if* $n > 1$*, to denote the parent path of* P.
- $\text{last}(P) = l_n$ *to denote the final label in* P.

For example, if $\mathbf{E} = \{\rho, \texttt{Phone}\}$ and $\mathbf{A} = \{\texttt{code}, \texttt{number}\}$ then code.Phone is a path but not a legal one, whereas ρ.Phone.code is a legal path. Also, ρ.Phone $\subseteq$ ρ.Phone.code and if $P = \rho$.Phone.code and $P' = \rho$.Phone.number, then $P \cap P' = \rho$.Phone.

We now define a path instance, which is essentially a downward sequence of nodes in an XML tree emanating from the root node.

Definition 3. *A* path instance $p = v_1.\cdots.v_n$ *is a non-empty sequence of nodes such that* $v_1 = v_\rho$ *and* $\forall i \in [2, n], v_{i-1} = \text{parent}(v_i)$*. The path instance* p *is said to be* defined over *a path* $P = l_1.\cdots.l_n$*, if* $\text{lab}(v_i) = l_i\ \forall i \in [1, n]$.

For example, referring to Fig. 3, $v_\rho.v_1.v_2$ is a path instance and it is defined over the path ρ.Customer.Phone.

The next definition specifies the set of nodes reachable in a tree from the root node by following a path.

Definition 4. *Given a tree* $\mathbb{T} = (\mathbf{L}, \mathbf{V}, E, \text{lab}, \text{val}, v_\rho)$ *and a path* P*, the function* $\text{N}(P)$ *returns the set of nodes defined by* $\text{N}(P) = \{v \in \mathbf{V} \mid v$ *is the final node in path instance* p *and* p *is defined over* $P\}$.

[1] $[1, n]$ denotes the set $\{1, \ldots, n\}$.

For instance, referring to Fig. 3, if $P = \rho.\texttt{Customer.cno}$, then $\mathrm{N}(P) = \{v_5, v_{13}\}$. We note that it follows from our tree model that for every node v in a tree $\mathbb{T}$ there is exactly one path instance p such that v is the final node in p and therefore $\mathrm{N}(P) \cap \mathrm{N}(\bar{P}) = \emptyset$ if $P \neq \bar{P}$. We therefore say that P is *the* path such that $v \in \mathrm{N}(P)$.

3 Defining an XML Key

In this section we present the syntax and semantics of our definition of an XKey, starting with the syntax. As noted previously, our syntactic framework is the same as that used in specifying `key` and `unique` constraints in XML Schema, and the framework used in [5] for specifying XML keys.

Definition 5. *An XKey is a statement of the form* $(T, (P_1, \ldots, P_n))$*, where* T *is a path called the* selector*, and* $P_1, \ldots, P_n$ *are paths called* fields*, such that for all* $i \in [1, n]$*,* $T.P_i$ *is a legal path that ends in an attribute or text label.*

We now compare this definition to the key constraint in XML Schema, which is the basis for the syntax of an XKey. (i) We only consider simple paths in the selectors and fields, whereas XML Schema allows for a restricted form of XPath expressions. (ii) In contrast to an XKey, an XML Schema key also allows for relative constraints, whereby the key constraint is only evaluated in part of the XML tree. (iii) The restrictions on fields means that we only consider the identification of selector nodes by text/attribute nodes, whereas the key constraint in XML Schema also allows for field nodes being element nodes.

We should mention that restrictions (i) - (iii) are not intrinsic to our approach, and Definition 5 can easily be extended to handle these extension. Our reason for not considering these extensions here is so that we can concentrate on the main contribution of our paper, which is to apply different semantics to an XKey so as to capture the identification and uniqueness property of a key.

To define the semantics of an XKey, we present first a definition, originally presented in [6], which is central to our approach and is what distinguishes it from other approaches. The intuition behind it is as follows. In defining relational integrity constraints, it is implicit that the relevant data values belong to the same tuple. The following *closest* definition extends this property of two data values belonging to the same tuple to XML, that is if two nodes in the tree satisfy the *closest* property, then 'they belong to the same tuple'.

Definition 6. *Given nodes* v_1 *and* v_2 *in an XML tree* $\mathbb{T}$*, the boolean function* $\mathrm{closest}(v_1, v_2)$ *is defined to return* true*, iff there exists a node* v_2^1 *such that (i)* $v_2^1 \in \mathrm{aancestor}(v_1)$*, and (ii)* $v_2^1 \in \mathrm{aancestor}(v_2)$*, and (iii)* $v_2^1 \in \mathrm{N}(P_1 \cap P_2)$*, where* P_1 *and* P_2 *are the paths such that* $v_1 \in \mathrm{N}(P_1)$ *and* $v_2 \in \mathrm{N}(P_2)$ *and the* aancestor *function is defined by* $\mathrm{aancestor}(v) = \mathrm{ancestor}(v) \cup \{v\}$.

For instance in the tree in Fig. 3, $\mathrm{closest}(v_2, v_5)$ is true since $v_2 \in \mathrm{N}(\rho.\texttt{Customer.Phone})$, $v_5 \in \mathrm{N}(\rho.\texttt{Customer.cno})$ and $v_1 \in \mathrm{N}(\rho.\texttt{Customer})$ is an *aancestor*

of both v_2 and v_5, where $\rho.\texttt{Customer} = \rho.\texttt{Customer.Phone} \cap \rho.\texttt{Customer.cno}$. Also, closest$(v_1, v_5)$ is true since v_1 is an *aancestor* of both v_1 and v_5 and $\rho.\texttt{Customer} = \rho.\texttt{Customer} \cap \rho.\texttt{Customer.cno}$. However, closest$(v_1, v_{13})$ is false since $v_{13} \in \mathrm{N}(\rho.\texttt{Customer.cno})$, but v_1 and v_{13} have no common *aancestor* node in $\mathrm{N}(\rho.\texttt{Customer})$.

This leads to the definition of the semantics of an XKey.

Definition 7. *An XML tree* $\mathbb{T}$ satisfies *an XKey* $\sigma = (T, (P_1, \ldots, P_n))$*, denoted by* $\mathbb{T} \vDash \sigma$*, iff whenever there exist selector nodes* $\{v, v'\} \subseteq \mathrm{N}(T)$ *and sets of field nodes* $v_1, \ldots, v_n$ *and* $v'_1, \ldots, v'_n$ *such that*

i) $\forall i \in [1, n]$, $\{v_i, v'_i\} \subseteq \mathrm{N}(T.P_i)$ *and* $v_i \in \mathrm{ancestor}(v)$ *and* $v'_i \in \mathrm{ancestor}(v')$*;*
ii) $\forall i, j \in [1, n]$, $\mathrm{closest}(v_i, v_j) = \mathrm{closest}(v'_i, v'_j) = \mathrm{true}$*;*
iii) $\forall i \in [1, n], \mathrm{val}(v_i) = \mathrm{val}(v'_i)$,
then $\forall i \in [1, n], v_i = v'_i$.

Clearly, our definition of an XKey captures the uniqueness property of the sets of field nodes in a key σ, since if $\mathbb{T} \vDash \sigma$ then there cannot exist two distinct sets of field nodes for σ that are value equal. Also, our definition captures the identification of selector nodes by the combination of field nodes. That is, if $\mathbb{T} \vDash \sigma$ and v, v' are selector nodes for σ, then $v = v'$ if there exist field nodes $v_1, \ldots, v_n$ for v and $v'_1, \ldots, v'_n$ for v' such that $\mathrm{val}(v_i) = \mathrm{val}(v'_i)$ for all $i \in [1, n]$. This is because if $v \neq v'$, then since $\mathbb{T}$ is a tree a field node cannot be a descendant of both v and v' and hence the sets of field nodes $v_1, \ldots, v_n$ and $v'_1, \ldots, v'_n$ must be distinct, which is a contradiction and so $v = v'$.

For instance, if $\bar{\mathbb{T}}$ is the tree in Fig. 3, then nodes v_1 and v_9 are the selector nodes for the XKey $\kappa' = (\rho.\texttt{Customer}, (\texttt{Phone.code}, \texttt{Phone.number}))$ in tree $\bar{\mathbb{T}}$, and the only sets of field nodes that pairwise satisfy the *closest* property are $\{\{v_3, v_4\}, \{v_7, v_8\}\}$ for v_1 and $\{\{v_{11}, v_{12}\}, \{v_{15}, v_{16}\}\}$ for v_9. These sets of field nodes are obviously unique, since none of them are value equal, and therefore tree $\bar{\mathbb{T}}$ satisfies κ'. Note that the selector nodes v_1 and v_9 are indeed identified by each of the sets of field nodes $\{v_3, v_4\}, \{v_7, v_8\}$ and $\{v_{11}, v_{12}\}, \{v_{15}, v_{16}\}$, respectively.

Now, recall from the introductory example that $\bar{\mathbb{T}}$ is the tree obtained from relation `Phones` (cf. Fig. 2a) according to the general mapping procedure originally presented in [6], whereby flat relation `Phones` is first nested on $\{\texttt{code}, \texttt{number}\}$ and then mapped directly to XML. In this example $(\texttt{code}, \texttt{number})$ is the key for `Phones`, and its semantics is preserved by the XKey κ' in the obtained tree $\bar{\mathbb{T}}$ since $\bar{\mathbb{T}} \vDash \kappa'$ as shown above. Hence κ' exemplifies the property of an XKey to preserve the semantics of a relational key in case that the XML document is obtained from a complete relation by the mapping procedure in [6]. We omit a formal analysis of this property of an XKey here for reasons of space requirements and refer the reader to [9] instead.

We note finally that κ' is violated in tree $\bar{\mathbb{T}}$ according to both the semantics of [5] and XML Schema, which shows that these approaches do not have the desirable property of preserving the semantics of a relational key.

4 Related Work and Discussion

In recent years, several types of XML Integrity Constraints (XICs) have been studied. We focus here on key and related types of constraints and refer the reader to [4] for a survey of other types of XICs.

Related to keys in XML are functional dependencies in XML (XFDs), which have been proposed using a 'tree tuple' approach [10] and a 'closest node' approach [6][2]. Tailored to the selector/field framework, an XFD achieves the identification of selector nodes in the same, sophisticated way an XKey does. It however does not account for the uniqueness of field nodes. This together with the limitations of the well recognized proposals towards XML keys in [5] and the XML Schema specification, which we have illustrated throughout the paper, in fact motivated the definition and study of an XKey.

The XML key presented in [11] does not use the selector/field framework, but instead identifies element nodes by a subset of their attributes. The targeted element nodes are thereby designated by either a type in a DTD or by a path expression. This approach is less expressive than an XKey since it does not allow for the identification of element nodes by descending field nodes.

In future work we will investigate the consistency and implication problems for XKeys, which are fundamental to any type of integrity constraint. These problems are formulated as the questions of whether there exists at least one XML document that satisfies a given set of XKeys, and whether a single XKey must hold in an XML document given that a set of XKeys holds, respectively.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Möller, A., Schwartzbach, M.: An Introduction to XML and Web Technologies. Addison Wesley, Reading (2006)
3. Beyer, K.S., Cochrane, R., Josifovski, V., Kleewein, J., Lapis, G., Lohman, G.M., Lyle, R., Özcan, F., Pirahesh, H., Seemann, N., Truong, T.C., der Linden, B.V., Vickery, B., Zhang, C.: System RX: One Part Relational, One Part XML. In: SIGMOD, pp. 347–358. ACM, New York (2005)
4. Fan, W.: XML Constraints: Specification, Analysis, and Applications. In: DEXA Workshops, pp. 805–809. IEEE Computer Society Press, Los Alamitos (2005)
5. Buneman, P., Davidson, S.B., Fan, W., Hara, C.S., Tan, W.C.: Reasoning about keys for XML. Information Systems 28(8), 1037–1063 (2003)
6. Vincent, M.W., Liu, J., Mohania, M.: On the Equivalence between FDs in XML and FDs in Relations. Acta Informatica 44(3-4), 207–247 (2007)
7. Fan, W.: XML Publishing: Bridging Theory and Practice. In: Arenas, M., Schwartzbach, M.I. (eds.) DBPL 2007. LNCS, vol. 4797, pp. 1–16. Springer, Heidelberg (2007)
8. Vakali, A., Catania, B., Maddalena, A.: XML Data Stores: Emerging Practices. IEEE Internet Computing 9(2), 62–69 (2005)

[2] These two approaches have been shown to be equivalent in complete XML trees [6].

9. Karlinger, M., Vincent, M., Schrefl, M.: Keys for XML. Technical Report 09.03, Dept. of Business Informatics - DKE, JKU Linz (2009)
10. Arenas, M., Libkin, L.: An information-theoretic approach to normal forms for relational and XML data. J. ACM 52(2), 246–283 (2005)
11. Arenas, M., Fan, W., Libkin, L.: On the Complexity of Verifying Consistency of XML Specifications. SIAM J. Comput. 38(3), 841–880 (2008)