

Expressive, yet Tractable XML Keys

Sven Hartmann
Department of Informatics
Clausthal University of Technology, Germany
Sven.Hartmann@tu-clausthal.de

Sebastian Link
School of Information Management
Victoria University of Wellington, New Zealand
Sebastian.Link@vuw.ac.nz

ABSTRACT

Constraints are important for a variety of XML recommendations and applications. Consequently, there are numerous opportunities for advancing the treatment of XML semantics. In particular, suitable notions of keys will enhance XML's capabilities of modeling, managing and processing native XML data. However, the different ways of accessing and comparing XML elements make it challenging to balance expressiveness and tractability.

We investigate XML keys which uniquely identify XML elements based on a very general notion of value-equality: isomorphic subtrees with the identity on data values. Previously, an XML key fragment has been recognised that is robust in the sense that its implication problem can be expressed as the reachability problem in a suitable digraph. We analyse the impact of extending this fragment by structural keys that uniquely identify XML elements independently of any data. We establish a sound and complete set of inference rules for this expressive fragment of XML keys, and encode these rules in an algorithm that decides the associated implication problem in time quadratic in the size of the input keys. Consequently, we gain significant expressiveness without any loss of efficiency in comparison to less expressive XML key fragments.

1. INTRODUCTION

The eXtensible Markup Language (XML,[7]) provides a high degree of flexibility to create and mark-up data. Consequently, many users find it very simple to produce XML documents. As a result XML has become the standard to exchange and integrate data on the Web and elsewhere. There has been wide consensus among researchers and practitioners that XML requires commensurate data management tools that can store, manipulate, and process XML data in its native format. However, the syntactic flexibility and complex tree-like nested data make it challenging to express desirable properties of XML data and provide facilities that can reason efficiently about such properties.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the ACM. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permissions from the publisher, ACM. *EDBT 2009*, March 24–26, 2009, Saint Petersburg, Russia. Copyright 2009 ACM 978-1-60558-422-5/09/0003 ...\$5.00

XML constraints restrict XML documents to those considered meaningful to the application domain under consideration. Apart from the ability to express fundamental properties of data their significance is due to a wide range of applications ranging from schema design, query optimisation, efficient storing and updating, data exchange and integration, to data cleaning [15].

One of the most fundamental classes of XML constraints are keys. While there is a well-understood and fully accepted notion of a key in relational databases, there are different popular proposals for the notion of an XML key. It is quite likely that there will never be a best notion. Instead, database designers and administrators will choose a suitable notion from a whole pool of different XML key concepts. Such a choice may be based on the preference to express desirable properties, guarantee efficient reasoning about keys, or the support necessary for specific XML applications.

1.1 Previous Work

As it is the case with many other classes of constraints it is a major challenge to find natural and useful notions of XML keys whose associated decision problems are also tractable [15, 16, 17, 31, 33]. For example, XML Schema allows to specify certain keys but the associated consistency problem has been shown to be intractable for many subclasses [3]. Moreover, the specification of such keys requires the presence of a schema definition which is not mandatory for XML documents.

An alternative promising approach are the classes of absolute and relative keys as defined and studied by Buneman et al [8, 9]. Their definitions are independent of any schema formalism such as a DTD [7] or XSD [32]. Moreover, a robust fragment of these keys is known whose associated implication problem can be decided in time quadratic in the size of the input keys [21]. Absolute and relative keys are based on the representation of XML data as trees. This representation is commonly used by DOM [2], XPath [24], and XML Schema [32]. Figure 1 shows such a representation in which nodes are annotated by their type: E for element, A for attribute, and S for string (PCDATA).

More precisely, keys are expressions of the form

$$(Q, (Q', \{P_1, \dots, P_k\}))$$

in which Q, Q', P_1, \dots, P_k are node selection queries defined in terms of suitable XPath expressions. For a fixed XML tree T let $n[E]$ denote the set of nodes in T reachable from the node n of T by following the XPath expression E . The context path Q selects the set $r[Q]$ of context nodes, where r denotes the root node of T . For each context node $u \in r[Q]$

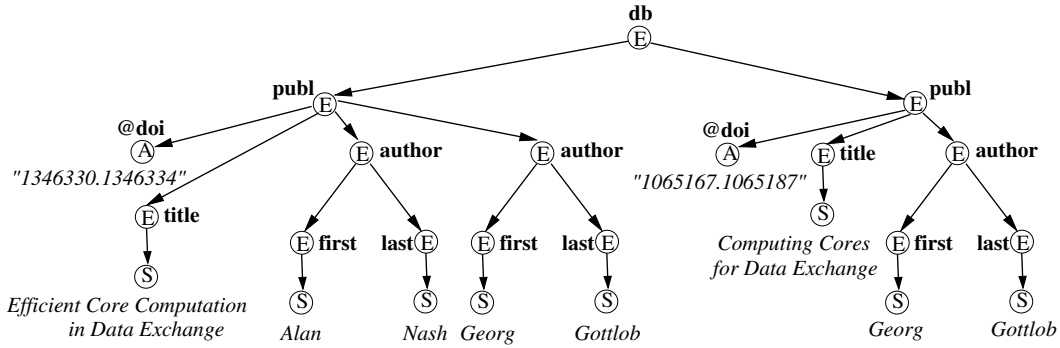


Figure 1: An XML data tree

the target path Q' selects the set $u[Q']$ of target nodes, and for every target node $v \in u[Q']$ and for every $i = 1, \dots, k$ the key path P_i selects the set $v[P_i]$ of key nodes. An XML tree satisfies the key $(Q, (Q', \{P_1, \dots, P_k\}))$ if for all target nodes v, v' that belong to the same set of context nodes the following holds: if for all $i = 1, \dots, k$ the key node sets $v[P_i]$ and $v'[P_i]$ agree, then v and v' agree themselves. Different XML key proposals define different notions of agreement. For a comparison of popular XML key proposals see [18]. In this paper we follow the notion of agreement from [8, 9, 21] where $v[P_i]$ and $v'[P_i]$ agree whenever there are nodes $w \in v[P_i]$ and $w' \in v'[P_i]$ such that w and w' form the roots of isomorphic subtrees with the identity on strings. Moreover, v and v' agree in the definition above whenever they denote the same node.

In Figure 1 the following absolute key is reasonable: the *doi*-value identifies the *publ* node in the entire tree. 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. These examples already indicate that the choice of an XPath language for accessing context nodes, target nodes and key nodes is crucial. If the language is too rich many properties may be expressed but reasoning may become intractable, or even infeasible. If the language is too simple, reasoning may be extremely efficient but the language is incapable of expressing many important properties of the application domain. For the original notion of XML keys [8] the context and target path expressions may use parent-child and ancestor-descendant navigation as well as wildcards for node labels, while key path expressions may use parent-child navigation. In [9] context and target path expressions may not use node label wildcards, but key path expressions may use both parent-child and ancestor-descendant navigation. So far, however, an axiomatisation only exists for the fragment of XML keys from [9] in which the set of key path expressions is required to be non-empty and key path expressions may use parent-child navigation [21]. The implication problem of this fragment is characterised by the reachability problem of a suitable digraph and decidable in time quadratic in the size of the input [20]. The axiomatisability and implication problem of most other XML key fragments are still open [8]. In this paper, we will

analyse the impact of keys with an empty set of key path expressions on the fragment studied in [20].

Interestingly, an empty set of key path expressions can add a significant amount of expressiveness to the language of XML keys. More precisely, the previous examples of XML keys have all identified nodes based on the values of the key nodes. In sharp contrast, *structural keys*, i.e. keys with an empty set of key path expressions, identify nodes independently of any data. For instance, there is at most one *doi* subnode in every subtree rooted at a *publ* node. In other words, there is at most one document identifier for each publication. On the other hand, *publ* nodes may have more than a single *author* descendant unless we want to restrict our attention to publications authored by a single person. Hence, structural keys can express significant properties of XML data, but identify nodes quite differently from previously studied keys.

Notice that the articles in [12, 13] analyse an orthogonal fragment of XML keys where value-equality is restricted to string-equality on attribute nodes but structural keys are permitted. This semantics deviates considerably from the more expressive approach, well grounded by the same authors in their earlier papers [8, 9]. Our work here carries forward the investigation of the original approach. Indeed, we will focus on the interaction between structural keys and the keys investigated in [21].

More recent work on XML constraints include [4, 5, 8, 9, 10, 11, 16, 17, 20, 21, 22, 35, 37], for a brief survey see [15].

1.2 Contributions

We investigate the XML key fragment \mathcal{K}' that results from extending the robust fragment \mathcal{K} from [21] by structural keys. We believe that an investigation of this fragment is important for several reasons:

- as the name suggests, structural keys provide a means to identify elements of an XML document based on the structure of the document alone and independently of any data. This is a feature that distinguishes XML from other data models, and therefore justifies a special investigation.
- Structural keys occur naturally in XML documents. For instance, these keys can express the uniqueness of attribute labels as required by the W3C recommendation [7]. Moreover, among other constraints structural keys also cover the *maxOccurs*-attribute provided by XML Schema [32] where the value is 1.

- Structural keys identify element nodes by means that are different from classes that have previously been studied. Therefore, it is interesting to investigate their impact on such classes.
- Apart from node label wildcards the class \mathcal{K}' defines exactly those XML keys that were originally proposed in [8].
- Our results show that \mathcal{K}' forms a large tractable fragment of numerical constraints whose implication problem is strongly *coNP*-hard. Hence, \mathcal{K}' constitutes a class of XML keys that is attractive to many XML applications.

In contrast to the class \mathcal{K} we cannot characterise the implication of keys in \mathcal{K}' in terms of the reachability problem in a suitable digraph. This is not a surprise because structural keys do use different means to identify element nodes.

Next we characterise the expressiveness of \mathcal{K}' in purely syntactical terms. More precisely, we establish an axiomatisation for \mathcal{K}' , i.e., a set of inference rules which is sound and complete for the reasoning about XML keys in \mathcal{K}' . The axiomatisation has several benefits:

- one can infer all implicitly specified keys from the explicitly specified keys. This means we have an automatic way of exhausting all the conclusions that follow from the specification of our keys. Such a method is extremely handy for design and validation purposes, for data cleaning, and also for query optimisation. In particular, we can ensure that we have exploited all opportunities of utilising implicit knowledge for these purposes.
- it provides insight into the properties of structural keys. In contrast to the fragment \mathcal{K} , for instance, in \mathcal{K}' we have non-trivial absolute keys that are consequences of relative keys.
- the syntactic inference rules can be encoded in an algorithm that decides the implication of keys in \mathcal{K}' efficiently.

Indeed, we will devise an algorithm that decides XML key implication in \mathcal{K}' in time quadratic in the size of the input keys. Notice that this algorithm requires as input a set of explicitly specified XML keys and an additional XML key and returns yes precisely when this additional key is implied by the set of explicitly specified keys. An enumeration algorithm based on the inference rules of an axiomatisation, however, requires only the set of explicitly specified keys as input, and returns an enumeration of all implicitly specified keys. Hence, these two algorithms establish different facilities to reason about XML keys. To the best of our knowledge, these are the first reasoning facilities for the fragment \mathcal{K}' . Our results show that there is no loss in efficiency when adding structural keys to \mathcal{K} . Consequently, our algorithms establish a gain in expressiveness that comes for free in terms of efficiency.

In contrast, it has been shown that the implication of *upper-bound constraints*, which allow the specification of arbitrary upper bounds on the number of target nodes (e.g. that each author has at most three first names), is *strongly coNP*-hard to decide. Therefore, our results show that the

fragment \mathcal{K}' (in which keys have a fixed upper bound of 1) forms an expressive tractable subclass [20].

It is not surprising that the difference in the semantics of keys in [12, 13] to keys in this and other papers [8, 9, 21] results in different axiomatisations and decision algorithms, cf. [13].

1.3 Organisation

We use Section 2 to formalise the underlying XML tree model, the navigational path languages, the notion of value equality and the notion of XML keys [8, 9]. In Section 3 we introduce the key fragment \mathcal{K}' , and establish a finite axiomatisation for the implication of keys in \mathcal{K}' . In Section 4 we briefly summarise the techniques that have previously been established for reasoning about keys in \mathcal{K} . Section 5 illustrates why these techniques cannot simply be extended to the larger key fragment \mathcal{K}' . Subsequently, we develop new techniques in Section 6 that enable us to reason about XML keys in \mathcal{K}' efficiently. It is demonstrated in Section 7 that \mathcal{K}' forms a large tractable subclass of another constraint class which is likely to be intractable. Finally, we conclude and briefly comment on future work in Section 8.

2. PREREQUISITES

In this section we recall the basics of the XML tree model, the notion of value equality, and describe the path language used to locate sets of nodes within an XML tree. Throughout the paper we assume familiarity with basic terminology from graph theory, see e.g. [23].

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

An *XML tree* is a 6-tuple $T = (V, lab, ele, att, val, r)$ where V denotes a set of nodes, and lab is a mapping $V \rightarrow \mathcal{L}$ assigning a label to every node in V . A node $v \in V$ is called an *element node* if $lab(v) \in \mathbf{E}$, an *attribute node* if $lab(v) \in \mathbf{A}$, and a *text node* if $lab(v) = S$. Moreover, 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. The partial mapping val assigns a string to each attribute and text node: for each node $v \in V$, $val(v)$ is a string if v is an attribute or text node, while $val(v)$ is undefined otherwise. Finally, r is the unique and distinguished root node. T is said to be *finite* if V is finite, and is said to be *empty* if V consists of the root node only.

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 . A *path* p of T is a finite sequence of nodes v_0, \dots, v_m in V such that (v_{i-1}, v_i) is an edge of T for $i = 1, \dots, m$. We call p a path from v_0 to v_m , and say that v_m is reachable from v_0 following the path p . The path p determines a word $lab(v_1) \dots lab(v_m)$ over the alphabet \mathcal{L} , denoted by $lab(p)$. For a node $v \in V$, each node w reachable from v is called a *descendant* of v . Note that an XML tree has a tree structure: for each node $v \in V$, there is a unique path from the root node r to v .

We can now define value equality for pairs of nodes in an XML tree. 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 $i = 1, \dots, k$.

Note that the notion of value equality takes the document order of the XML tree into account. As an example, all *teacher*-nodes in Figure 1 are value equal. We remark that $=_v$ is an equivalence relation on the node set V of the XML tree. This is easy to observe as value equality between nodes corresponds to isomorphism of the subtrees rooted at these nodes.

In order to define XML keys we need a mechanism for selecting nodes in an XML tree. Path expressions have been widely used for node selection in XML theory and practice, cf. [24, 31]. We are interested in path languages that are expressive enough to be practical, yet sufficiently simple to be reasoned about efficiently. This is the case for the languages PL and PL_s that have been used in [8, 9] for the definition of XML keys. For the sake of completeness we will briefly introduce these languages here.

Let $_*$ be a distinguished symbol not in \mathcal{L} . It will serve as a *variable length don't care* wildcard, that is, as a combination of a *single symbol* wildcard (denoted by $_$) and the Kleene star ($*$). Let PL denote the set of all words over the alphabet $\mathcal{L} \cup \{_*\}$. Further let PL_s be the subset of PL containing all words over the alphabet \mathcal{L} . Both PL and PL_s form free monoids with the binary operation of concatenation (denoted by \cdot) and the empty word (denoted by ε) as identity element.

Let P, Q be words from PL . P is a *refinement* of Q , denoted by $P \in Q$, if P is obtained from Q by replacing wildcards in Q by words from PL . For example, *publ.author.first* is a refinement of $_*.first$. Note that \in is a pre-order on PL . Let \sim denote the congruence induced by the identity $_*._* = *_*$ on PL . Observe that $P \sim Q$ holds if and only if P and Q are refinements of each other.

We now define the semantics of words from PL in the context of XML. Let Q be a word from PL . A path p in the XML tree T is called a Q -path if $lab(p)$ is a refinement of Q . For nodes $v, w \in V$, we write $T \models Q(v, w)$ if w is reachable from v following a Q -path in T . For example, in the XML tree in Figure 1, all *first*-nodes are reachable from the root node following a *publ.author.first*-path. Obviously, they are also reachable from the root node following

a $_*.first$ -path. For a node $v \in V$, let $v[[Q]]$ denote the set of nodes in T that are reachable from v following any Q -path, that is, $v[[Q]] = \{w \mid T \models Q(v, w)\}$. As an example consider the second *semester*-node v in the XML tree in Figure 1. Then $v[[_*.student]]$ is the set of all *student*-nodes that are descendants of the second *semester* node. We use $[[Q]]$ as an abbreviation for $r[[Q]]$ where r is the root node of T . Thus, $[[_*.first]]$ is the set of all *first*-nodes in the entire XML tree.

Recall that each attribute or text node in an XML tree T is a leaf. Therefore, a word Q from PL is said to be *valid* if it does not have labels $\ell \in \mathbf{A}$ or $\ell = S$ in a position other than the last one. Note that each prefix of a valid Q is valid, too.

Let P, Q be words from PL . P is *contained* in Q , denoted by $P \subseteq Q$, if for every XML tree T and every node v of T we have $v[[P]] \subseteq v[[Q]]$. It follows immediately from the definition that $P \in Q$ implies $P \subseteq Q$.

The containment problem of PL is to decide, given valid P and Q from PL , whether $P \subseteq Q$ holds. In [9] it is shown that valid P, Q from PL satisfy $P \subseteq Q$ if and only if P is a refinement of Q and that the containment problem of PL can be decided in quadratic time.

In accordance with [8] we will work with the quotient set $PL_{/\sim}$ rather than with PL directly: A word from PL is in *normal form* if it has no consecutive wildcards. Each congruence class contains a unique word in normal form. Each word from PL can be transformed into normal form in linear time, just by removing superfluous wildcards. In particular, each word from PL_s is in normal form. The *length* $|Q|$ of a PL expression Q is the number of labels in Q plus the number of $_*$ in the normal form of Q , cf. [9]. The empty path expression ε has length 0. The natural homomorphism from PL to $PL_{/\sim}$ is an isomorphism when restricted to words in normal form. By abuse of notation we will use the words from PL to denote their respective congruence class, cf. [8]. It is a straightforward exercise to extend the terminology introduced above for PL to $PL_{/\sim}$.

We will call members of $PL_{/\sim}$ (and $PL_{s/\sim}$) *PL expressions* (or *PL_s expressions*, respectively) in order to emphasise their use for node selection in XML. Note that there is an easy conversion of PL expressions to XPath expressions [24], just by replacing “ $_*$ ” with “ $./$ ” and “ $_$ ” with “ $/$ ”.

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

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

3. KEYS FOR XML

In this paper we study the following fragment \mathcal{K}' of XML keys.

DEFINITION 1. *A key constraint φ for XML (or short XML key) is an expression $(Q, (Q', S))$ where Q, Q' are PL expressions and S is a finite set of PL_s expressions such that $Q.Q'.P$ are valid PL expressions for all P in S , if S*

is non-empty, and $Q.Q'$ is a valid PL expression, if S is empty. Herein, Q is called the context path, Q' is called the target path, and the elements of S are called the key paths of φ . If $Q = \varepsilon$, we call φ an absolute key; otherwise φ is called a relative key. \square

For an XML key φ , we use Q_φ to denote its context path, Q'_φ to denote its target path, S_φ to denote its set of key paths and $P_1^\varphi, \dots, P_{k_\varphi}^\varphi$ to denote the elements of S_φ , where k_φ is the non-negative number of its key paths.

Finally, an XML key φ is called *structural* if and only if $S_\varphi = \emptyset$, and let \mathcal{K} denote the class of XML keys that results from \mathcal{K}' by removing all structural keys.

DEFINITION 2. An XML tree T satisfies the XML key φ if and only if for all nodes $q \in \llbracket Q_\varphi \rrbracket$ and all nodes $q'_1, q'_2 \in q \llbracket Q'_\varphi \rrbracket$ such that for all $i = 1, \dots, k$ there are nodes $x_i \in q'_1 \llbracket P_i^\varphi \rrbracket, y_i \in q'_2 \llbracket P_i^\varphi \rrbracket$ with $x_i =_v y_i$, then $q'_1 = q'_2$. \square

In particular, an XML tree satisfies the structural key φ if and only if for all nodes $q \in \llbracket Q_\varphi \rrbracket$ the set $q \llbracket Q'_\varphi \rrbracket$ contains at most one element.

EXAMPLE 1. We formalise some of the examples from the introduction. In an XML tree that satisfies the absolute key

$$(\varepsilon, (-^*.publ, \{doi\}))$$

one will never be able to find two different publ nodes that have value equal doi subnodes. Furthermore, under a publ node in an XML tree that satisfies the relative key

$$(-^*.publ, (author, \{first.S, last.S\}))$$

one will never find two different author subnodes that are value-equal on some first.S subnodes and on some last.S subnodes.

If we specify the structural key

$$(-^*.publ, (doi, \emptyset)),$$

then every legal XML tree must not have any two distinct doi children nodes under any publ-node. \square

EXAMPLE 2. Definition 1 guarantees that keys in \mathcal{K}' , in particular structural keys, cover XML Schema constraints of the form `maxOccurs="1"` [32]. However, keys in the class \mathcal{K} cannot express these constraints. Moreover, structural keys can also express common constraints that cannot be expressed by the `maxOccurs` attribute. As an example, consider a university that is divided into several faculties. Each faculty has several schools, each school has a head, and for every faculty there is a single dean who is one of the heads in the schools of the faculty. Based on this format, we may then specify the structural key

$$(-^*.faculty, (school.head.dean, \emptyset))$$

stating that in every faculty there is at most one dean who is the head of some school within the faculty. \square

Let Σ be a finite set of keys in \mathcal{K}' . An XML tree T satisfies Σ if and only if T satisfies every $\sigma \in \Sigma$. Let $\Sigma \cup \{\varphi\}$ be a finite set of keys in \mathcal{K}' . We say that Σ (finitely) implies φ , denoted by $\Sigma \models_{(f)} \varphi$, if and only if every (finite) XML tree T that satisfies Σ also satisfies φ . The (finite) implication problem is to decide, given any finite set of keys $\Sigma \cup \{\varphi\}$, whether $\Sigma \models_{(f)} \varphi$. For a finite set Σ of keys in \mathcal{K}' , let

$\Sigma_{(f)}^* = \{\varphi \in \mathcal{K}' \mid \Sigma \models_{(f)} \varphi\}$ be its (finite) semantic closure, i.e., the set of all keys (finitely) implied by Σ . Finite and unrestricted implication problem coincide, even for a more general class of XML keys than \mathcal{K}' , cf. [9]. We will therefore commonly speak of the *implication problem* for keys in \mathcal{K}' .

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

As a first contribution we establish that the set \mathfrak{R} of inference rules in Table 1 is *sound* (i.e., $\Sigma_{\mathfrak{R}}^+ \subseteq \Sigma^*$) and *complete* (i.e., $\Sigma^* \subseteq \Sigma_{\mathfrak{R}}^+$) for the implication of keys in \mathcal{K}' .

Notice that the first nine inference rules of the top three lines in Table 1 form an axiomatisation for the class \mathcal{K} [21]. In that case, the *prefix* rule reduces to the *epsilon-prefix* rule

$$\frac{(Q, (Q', S \cup \{\varepsilon, P'\}))}{(Q, (Q', S \cup \{\varepsilon, P'.P''\}))}, \text{ cf. [21].}$$

We will exemplify the four new inference rules by examples from the XHTML language [29]. The examples will also provide insight into the expressiveness of structural keys.

Suppose that for each table row of an XHTML table a cell (table data) can be identified by its contents (the string S), and suppose further that in each XHTML table there is at most one table row. Then it follows that in each XHTML table, each cell of that row is uniquely identified by its contents. This reasoning can be formalised by our keys: the two keys

$$\sigma_1 = (-^*.table.tr, (td, \{S\}))$$

and

$$\sigma_2 = (-^*.table, (tr, \emptyset))$$

imply the key

$$\varphi = (-^*.table, (tr.td, \{S\})).$$

Indeed, φ can be inferred from σ_1 and σ_2 by a single application of the *context-to-target* rule.

For path expressions $Q \in PL$ and $P \in PL_s$, Q is said to have *border* P if and only if there are some path expressions $Q', Q'' \in PL$ such that $Q = P.Q'$ and $Q = Q''.P$. The expression Q is said to have the *proper border* P if and only if Q has border P and $Q \neq P$. For instance, the path expression `ol.li.ol.li.ol` has proper border `ol.li.ol`, and the path expression `li` has proper border ε , but ε has no proper border since its only border ε is not proper.

Suppose that every ordered list of an XHTML document can contain at most one unordered list as a subelement. Then it is also true, for instance, that each subelement of an unordered list nested inside another unordered list inside an ordered list can be identified by its list item. In terms of XML keys this reads as follows: the key

$$\sigma = (-^*.ol, (-^*.ul, \emptyset))$$

implies the key

$$\varphi = (\varepsilon, (-^*.ol._^*.ul._^*.ul._^*, \{li\})).$$

However, φ can be inferred from σ by a single application of the *border* rule. In particular, `ul._^*.ul` is a proper border of `ul._^*.ul`.

The previous examples illustrate how structural keys enable us to reason about patterns. For instance, certain nestings of elements may be restricted. Among others, our ax-

$\frac{}{(Q, (\varepsilon, S))}$ <small>(epsilon)</small>	$\frac{(Q, (Q', S \cup \{P, P.P'\})), (Q.Q', P, \emptyset)}{(Q, (Q', S \cup \{P, P.P'.P''\}))}$ <small>(prefix)</small>	$\frac{(Q, (Q', S))}{(Q, (Q', S \cup \{P\}))}$ <small>(superkey)</small>	$\frac{(Q, (Q'.P, \{P'\}))}{(Q, (Q', \{P.P'\}))}$ <small>(subnodes)</small>
$\frac{(Q, (Q', S))}{(Q'', (Q', S))} Q'' \subseteq Q$ <small>(context-path-containment)</small>	$\frac{(Q, (Q', S))}{(Q, (Q'', S))} Q'' \subseteq Q'$ <small>(target-path-containment)</small>	$\frac{(Q, (Q'.Q'', S))}{(Q.Q', (Q'', S))}$ <small>(target-to-context)</small>	$\frac{(Q, (Q'.P, \{\varepsilon, P'\}))}{(Q, (Q', \{\varepsilon, P.P'\}))}$ <small>(subnodes-epsilon)</small>
$\frac{(Q, (Q', \{P.P' \mid P' \in S\})), (Q.Q', (P, S))}{(Q, (Q'.P, S))}$ <small>(interaction)</small>	$\frac{(Q.Q', (Q'', S)), (Q, Q', \emptyset)}{(Q, (Q'.Q'', S))}$ <small>(context-to-target)</small>	$\frac{(Q, (-*.P, \emptyset))}{(\varepsilon, (Q.-*.Q'.-*, \emptyset))} \begin{array}{l} Q' \neq * \\ \text{has proper} \\ \text{border } P \end{array}$ <small>(border)</small>	$\frac{(Q, (Q'.P, \{P_1, \dots, P_k\})), (Q.Q', (P, \emptyset))}{(Q, (Q', \{P.P_1, \dots, P.P_k\}))}$ <small>(multiple subnodes)</small>
$\frac{(Q, (-*.P, \emptyset))}{(\varepsilon, (Q.-*, \{P_1.P.P', P_2.P.P''\}))} P_1 \neq P_2$ <small>(neighbour)</small>			

Table 1: An axiomatisation of XML keys in \mathcal{K}' .

iomatisation of XML keys establishes means to infer all implicitly specified restrictions of patterns from those that have been specified explicitly.

As another example consider the constraint that each definition list of an XHTML document contains at most one data definition. This restriction implies other restrictions, for instance that every subelement of a definition list can be uniquely identified by the data definition of a defining term and the data definition of a defining term of a further definition list. In terms of XML keys these restrictions read as follows: the XML key

$$\sigma = (-*.dl, (-*.dd, \emptyset))$$

implies the XML key

$$\varphi = (\varepsilon, (-*.dl.-*, \{dt.dd, dl.dt.dd\})).$$

Indeed, φ can be inferred from σ by a single application of the *neighbour* rule.

Finally, we illustrate the application of the *multiple subnodes* rule. Suppose that an XHTML document satisfies the restriction that every **form** element that occurs in any paragraph has at most one **input** field. Furthermore, the XHTML document satisfies the restriction that for each paragraph the input field of a form can be uniquely identified by its type and name. Then the XHTML document also satisfies the following constraint: for each paragraph the form element can be uniquely identified by its input type and its input name. Expressed as XML keys these requirements read as follows: the XML keys

$$\sigma_1 = (p.form, (input, \emptyset))$$

and

$$\sigma_2 = (p, (form.input, \{type, name\}))$$

imply the XML key

$$\varphi = (p, (form, \{input.type, input.name\})).$$

It is not difficult to see that φ can be inferred from σ_1 and σ_2 by a single application of the *multiple subnodes* rule.

Our first main result establishes that the inference rules from Table 1 form an axiomatisation for the class \mathcal{K}' of XML

keys. This provides us with an algorithm that enumerates all those XML keys in \mathcal{K}' that are implicitly specified by a set of explicitly specified keys. Such implicit knowledge may be utilised to validate user requirements, improve the design of the XML database, optimise and rewrite XPath and XQuery queries or perform data cleaning routines. Moreover, the enumeration algorithm guarantees that all avenues for any of these purposes can be explored, i.e., no implicitly specified XML key will be missed.

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

Furthermore, the tree model for XML adopted from [8] for our investigation here leaves considerable freedom to data designers. To some extent this flexibility can be exploited when constructing XML trees in the proofs. For certain applications one might want to incorporate additional features as specified by the W3C standard of XML. As an example we mention the uniqueness of attributes: no element may possess two distinct attribute children with the same label. This requirement cannot be captured by keys of the fragment \mathcal{K} . However, it may be expressed by the structural key $(-*, (l_a, \emptyset))$ with $l_a \in \mathbf{A}$.

Suppose we require the uniqueness of attributes as part of the XML tree model. Then the structural keys $(-*, (l_a, \emptyset))$ with $l_a \in \mathbf{A}$ hold trivially. When adding a corresponding axiom to the inference rules in Table 1 we obtain a result similar to Theorem 1.

4. PREVIOUS TECHNIQUES

Theorem 1 provides us with a reasoning facility that infers all implicitly specified knowledge. In practice, however, it often suffices to validate partial implicit knowledge. That means, one is interested in deciding the implication problem of XML keys, i.e., the problem of deciding whether for an arbitrary finite set $\Sigma \cup \{\varphi\}$ of keys in \mathcal{K}' the XML key φ is implied by the set Σ of explicitly specified XML keys. We will use this section to briefly summarise the techniques developed for efficiently deciding the implication problem of keys in \mathcal{K} [21]. In Section 6 we will extend these techniques to the class \mathcal{K}' .

Let $\Sigma \cup \{\varphi\}$ be a finite set of keys in \mathcal{K} . Let $\mathcal{L}_{\Sigma, \varphi}$ denote the set of all labels $\ell \in \mathcal{L}$ that occur in path expressions of keys in $\Sigma \cup \{\varphi\}$, and fix a label $\ell_0 \in \mathbf{E} - \mathcal{L}_{\Sigma, \varphi}$. Further, let O_φ and O'_φ be the PL_s expressions obtained from the PL expressions Q_φ and Q'_φ , respectively, when replacing each $_*$ by ℓ_0 .

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

The following definition enables us to capture those nodes of $T_{\Sigma, \varphi}$ which must be duplicated in a possible counter-example tree. The *marking* of the mini-tree $T_{\Sigma, \varphi}$ is a subset \mathcal{M} of the node set of $T_{\Sigma, \varphi}$: if for all $i = 1, \dots, k_\varphi$ we have $P_i^\varphi \neq \varepsilon$, then \mathcal{M} consists of the leaves of $T_{\Sigma, \varphi}$, and otherwise \mathcal{M} consists of all descendant-or-selfs of q'_φ in $T_{\Sigma, \varphi}$. The nodes in \mathcal{M} are said to be *marked*.

EXAMPLE 3. The left of Figure 2 shows the mini-tree $T_{\Sigma, \varphi}$ for the key

$$\varphi = (\varepsilon, (_* . \text{publ}, (\text{author}, \{\text{first}.S, \text{last}.S\})))$$

and some Σ , where library is the fixed label chosen from $\mathbf{E} - \mathcal{L}_{\Sigma, \varphi}$. The marking of the mini-tree consists of its leaves (emphasised by \times). \square

We use mini-trees to calculate the impact of a key in Σ on a possible counter-example tree for the implication of φ by Σ . To distinguish keys that have an impact from those that do not, we introduce the notion of *applicability*. A key σ is said to be *applicable* to φ if and only if there are nodes $w_\sigma \in \llbracket Q_\sigma \rrbracket$ and $w'_\sigma \in w_\sigma \llbracket Q'_\sigma \rrbracket$ in $T_{\Sigma, \varphi}$ such that $w'_\sigma \llbracket P_i^\sigma \rrbracket \cap \mathcal{M} \neq \emptyset$ for all $i = 1, \dots, k_\sigma$. We say that w_σ and w'_σ *witness* the applicability of σ to φ .

EXAMPLE 4. Let Σ consist of the two keys

$$\sigma_1 = (\varepsilon, (_* . \text{publ}, \{\text{doi}\}))$$

and

$$\sigma_2 = (_* . \text{publ}, (\text{author}, \{\text{first}.S, \text{last}.S\})),$$

and let

$$\varphi = (\varepsilon, (_* . \text{publ}. \text{author}, \{\text{first}.S, \text{last}.S\})).$$

We find that σ_1 is not applicable to φ , while σ_2 is indeed applicable to φ . \square

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

EXAMPLE 5. Let $\Sigma = \{\sigma_1, \sigma_2\}$ and φ be as in Example 4. The witness graph $G_{\Sigma, \varphi}$ is illustrated in the middle of Figure 2. It contains a witness edge arising from σ_2 . \square

The size $|\varphi|$ of a key φ is defined as the sum of the lengths of all path expressions in φ , i.e., $|\varphi| = |Q_\varphi| + |Q'_\varphi| + \sum_{i=1}^{k_\varphi} |P_i^\varphi|$. Furthermore, let $\|\Sigma\|$ denote the sum of the sizes $|\sigma|$ over all $\sigma \in \Sigma$.

LEMMA 1 ([21]). Let $\Sigma \cup \{\varphi\}$ be a finite set of XML keys in \mathcal{K} . The node q_φ is reachable from q'_φ in the witness graph $G_{\Sigma, \varphi}$ if and only if $\varphi \in \Sigma^+$. \square

EXAMPLE 6. Let $\Sigma = \{\sigma_1, \sigma_2\}$ and φ as in Example 4. Since φ is not implied by Σ the completeness proof of the axiomatisation for \mathcal{K} includes the description of a generation of a counter-example tree. An example of such a counter-example tree is illustrated in Figure 2. \square

Based on Lemma 1 we have established the following algorithm for deciding XML key implication in the fragment \mathcal{K} .

ALGORITHM 1 (XML Key-IMPLICATION IN \mathcal{K}).

Input: finite set $\Sigma \cup \{\varphi\}$ of XML keys in \mathcal{K}

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

Method:

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

THEOREM 2 ([21]). Let $\Sigma \cup \{\varphi\}$ be a finite set of XML keys in \mathcal{K} . Algorithm 1 decides the implication problem $\Sigma \models \varphi$ in time $\mathcal{O}(|\varphi| \times (\|\Sigma\| + |\varphi|))$. \square

5. NON-EXTENSIBILITY

We will briefly describe some *showcases* that illustrate how Algorithm 1 fails when structural keys are permitted as well, i.e. for XML keys in \mathcal{K}' .

Let $\Sigma_1 = \{\text{ul}, (_* . \text{ol}, \emptyset)\}$ and

$$\varphi_1 = (\varepsilon, (\text{ul}. _* . \text{ol}. \text{li}. \text{ol}, \{\text{li}.S\})).$$

The relative structural key in Σ requires each *legal* XHTML document to have at most one ordered list nested inside each unordered list (on the level following the root of the document). Among other restrictions, it is implicit that every such legal document cannot have any ordered list occurring as a list item of another ordered list which is nested inside an unordered list. Hence, φ_1 must also be satisfied by every XHTML document that satisfies Σ_1 . More formally, the soundness of the *border*, *target-path-containment* and *superkey* rule shows that Σ_1 implies φ_1 .

Let us consider the witness graph G_{Σ_1, φ_1} in the left of Figure 3. The node q_{φ_1} is not reachable from the node q'_{φ_1} . Consequently, Algorithm 1 returns the wrong result, and the generated alleged counterexample tree (according to the general construction in the completeness argument, cf. [21]) violates Σ_1 , see Figure 3. Consequently, Algorithm 1 does not work correctly for the class \mathcal{K}' .

Algorithm 1 also fails for instances in which keys can be derived by an application of the *neighbour* rule, e.g., $\Sigma_2 = \{\text{ol}, (_* . \text{ul}, \emptyset)\}$ and

$$\varphi_2 = (\varepsilon, (\text{ol}. _* . \{\text{li}. \text{ul}. S, \text{li}. \text{ol}. \text{li}. \text{ul}\})).$$

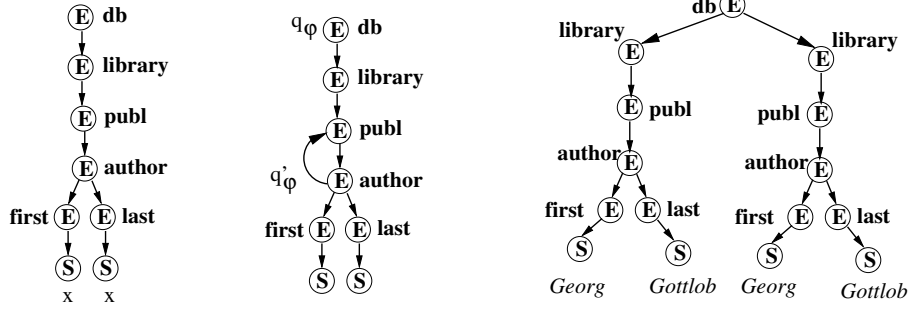


Figure 2: A mini-tree, witness graphs and counter-example tree.

In contrast to the case when structural keys are not permitted it now appears that non-trivial absolute keys are indeed implied by relative keys.

Let

$$\Sigma_3 = \{(\text{ol.li}, (\text{ol}, \emptyset)), (\text{ol}, (\text{li.ol}, \{\text{li.S}, \text{type}\}))\}$$

and

$$\varphi_3 = (\text{ol}, (\text{li}, \{\text{ol.li.S}, \text{ol.type}\})).$$

According to the soundness of the *multiple subnodes* rule Σ_3 implies φ_3 . However, q_{φ_3} is not reachable from q'_{φ_3} in the witness graph G_{Σ_3, φ_3} , cf. Figure 3. Consequently, Algorithm 1 returns the wrong result, and the generated alleged counterexample tree violates Σ_3 , cf. Figure 3.

6. XML KEY IMPLICATION

We have seen in the last section how the algorithm for deciding implication of keys in \mathcal{K} does not work for the broader class \mathcal{K}' . The showcases that highlight how the algorithm fails, however, have given us new insights into a possible extension of our techniques to the class \mathcal{K}' . We will now introduce and explain these extensions. In particular, we will establish an efficient algorithm for deciding implication of the fragment \mathcal{K}' . Indeed, the gain in expressiveness from \mathcal{K} to \mathcal{K}' comes for free in terms of efficiency.

Let $\Sigma \cup \{\varphi\}$ denote an instance of the implication problem for \mathcal{K}' where Σ contains some structural key. In this case, the mini-tree $T_{\Sigma, \varphi}$ may not be suitable for generating a possible counterexample tree because a single copy of the minitree may already violate some structural key in Σ that is applicable to φ .

As a first step we introduce tiny-trees as compressed versions of mini-trees. Intuitively, in the tiny-tree $\overline{T}_{\Sigma, \varphi}$ we identify separate paths in $T_{\Sigma, \varphi}$ that offend some structural key in Σ whenever we can, i.e., whenever such paths have the same sequence of node labels.

Two nodes v, w in the node set of the mini-tree $T_{\Sigma, \varphi}$ are said to be equivalent, denoted by $v \sim w$, if and only if there is some structural key $(Q_\sigma, (Q'_\sigma, \emptyset)) \in \Sigma^+$ such that there is some $w_\sigma \in \llbracket Q_\sigma \rrbracket$ and some simple path expression $P' \subseteq Q'_\sigma$ such that $v, w \in w_\sigma \llbracket P' \rrbracket$. The relation \sim is an equivalence relation on the node set of $T_{\Sigma, \varphi}$, i.e., reflexive, symmetric and transitive.

The tiny tree $\overline{T}_{\Sigma, \varphi}$ has as its node set the quotient of the node set of the mini-tree $T_{\Sigma, \varphi}$ with respect to \sim . The edge set of the tiny tree $\overline{T}_{\Sigma, \varphi}$ consists of precisely those edges (V, W) where $V \neq W$ and there are $v \in V$ and $w \in W$ such that (v, w) is an edge of the mini-tree $T_{\Sigma, \varphi}$.

ALGORITHM 2 (TINY-TREE GENERATION).

Input: mini-tree T

Output: tiny-tree \overline{T}

Method:

- (1) WHILE $\exists(Q_\sigma, (Q'_\sigma, \emptyset)) \in \Sigma$ such that
- (2) $\exists w_\sigma \in \llbracket Q_\sigma \rrbracket$ and $\exists P' \subseteq Q'_\sigma$ such that
 $\exists w'_\sigma, w''_\sigma \in w_\sigma \llbracket P' \rrbracket$ with $w'_\sigma \neq w''_\sigma$
- (3) DO Identify in T the paths from the root to w'' and w'
- (4) RETURN(T).

LEMMA 2. Algorithm 2 computes the tiny tree $\overline{T}_{\Sigma, \varphi}$ from the mini tree $T_{\Sigma, \varphi}$. \square

Let \mathcal{M} denote the marking of the mini tree $T_{\Sigma, \varphi}$. As a next step we define the marking $\overline{\mathcal{M}}$ of the tiny tree $\overline{T}_{\Sigma, \varphi}$ as follows: a node V of $\overline{T}_{\Sigma, \varphi}$ belongs to $\overline{\mathcal{M}}$ if and only if there is some $v \in V$ such that $v \in \mathcal{M}$ or there is some ancestor node W of V in $\overline{T}_{\Sigma, \varphi}$ that already belongs to $\overline{\mathcal{M}}$. Informally, the marking of the tiny tree is the downward closure of the marking of the mini tree with respect to document order.

EXAMPLE 7. Let $\Sigma = \{(\text{publ}, (\text{author}, \emptyset))\}$, and

$$\varphi = (\varepsilon, (\text{publ}, \{\text{author.first.S}, \text{author.last.S}\})).$$

The mini-tree $T_{\Sigma, \varphi}$, tiny-tree $\overline{T}_{\Sigma, \varphi}$ and a counterexample for the implication of φ by Σ are illustrated in Figure 4. \square

For the class \mathcal{K}' the tiny-tree $\overline{T}_{\Sigma, \varphi}$ will take on the role that the mini-tree $T_{\Sigma, \varphi}$ played for the class \mathcal{K} . In fact, the witness graph $G_{\Sigma, \varphi}$ is defined exactly as before but results from the tiny-tree $\overline{T}_{\Sigma, \varphi}$ rather than the mini-tree $T_{\Sigma, \varphi}$.

A key $\sigma \in \mathcal{K}'$ is said to be *applicable* to φ if and only if there are nodes $w_\sigma \in \llbracket Q_\sigma \rrbracket$ and $w'_\sigma \in w_\sigma \llbracket Q'_\sigma \rrbracket$ in $\overline{T}_{\Sigma, \varphi}$ such that $w'_\sigma \llbracket P^\sigma \rrbracket \cap \overline{\mathcal{M}} \neq \emptyset$ for all $P^\sigma \in S_\sigma$. We say that w_σ and w'_σ *witness* the applicability of σ to φ .

In order to decide implication in \mathcal{K}' we now only require a single additional step which results from the following lemma. Intuitively, if the tiny tree still violates some structural key in Σ , then there cannot be any counterexample for the implication of φ by Σ .

LEMMA 3. Let $\Sigma \cup \{\varphi\}$ be a finite set of XML keys in \mathcal{K}' , and let $\overline{T}_{\Sigma, \varphi}$ denote its tiny tree. If there is some structural key $\sigma \in \Sigma$ such that there is some $w_\sigma \in \llbracket Q_\sigma \rrbracket$ in $\overline{T}_{\Sigma, \varphi}$ and there are distinct $w'_\sigma, w''_\sigma \in w_\sigma \llbracket Q'_\sigma \rrbracket$ in $\overline{T}_{\Sigma, \varphi}$, then $\{\sigma\} \vdash \varphi$. \square

EXAMPLE 8. Let

$$\Sigma = \{(\text{publ}, (-^*.\text{author}, \emptyset))\},$$

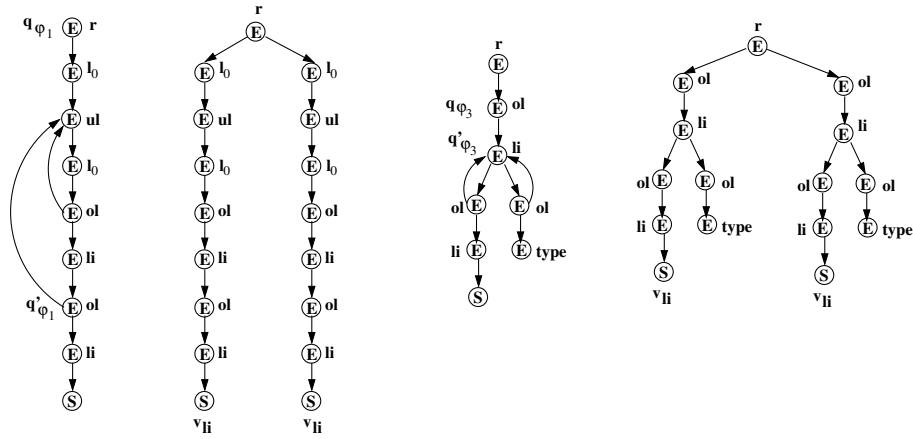


Figure 3: Witness graphs and the alleged counter-example tree.

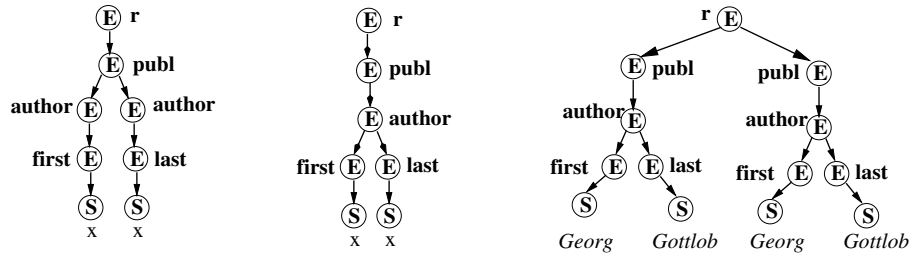


Figure 4: Mini-tree $T_{\Sigma, \varphi}$, tiny-tree $\bar{T}_{\Sigma, \varphi}$ and counterexample for Example 7.

and

$$\varphi = (\varepsilon, (\text{publ.}^*, \{\text{journal.author}, \text{conference.author}\})).$$

The mini-tree $T_{\Sigma, \varphi}$ is illustrated in Figure 5. Since there is no single simple path expression via which both of the distinct author nodes in $T_{\Sigma, \varphi}$ can be reached from the publ node, the tiny-tree coincides with the mini-tree. Notice that φ can be inferred from Σ by a single application of the neighbour rule. \square

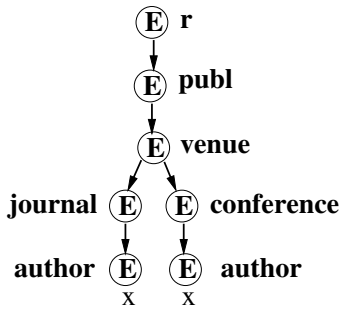


Figure 5: Mini-tree and Tiny-tree for Example 8.

Lemma 3 implies, in particular, that we cannot characterise key implication in \mathcal{K}' by reachability in the witness graph, as established for the class \mathcal{K} . However, reachability is still a sufficient condition. The proof encodes paths of the witness graph into inference steps to derive φ from Σ . This is accomplished by utilising correspondences between witness edges and applicable keys.

LEMMA 4. Let $\Sigma \cup \{\varphi\}$ be a finite set of XML keys in \mathcal{K}' . If q_φ is reachable from q'_φ in the witness graph $G_{\Sigma, \varphi}$, then $\varphi \in \Sigma^+$. \square

ALGORITHM 3 (XML Key-IMPLICATION IN \mathcal{K}').

Input: finite set $\Sigma \cup \{\varphi\}$ of XML keys in \mathcal{K}'

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

Method:

- (1) Construct $\bar{T}_{\Sigma, \varphi}$ from Σ and φ ;
- (2) **IF** $\exists (Q_\sigma, (Q'_\sigma, \emptyset)) \in \Sigma$ such that $\exists w_\sigma \in \llbracket Q_\sigma \rrbracket$ in $\bar{T}_{\Sigma, \varphi}$ and \exists distinct $w'_\sigma, w''_\sigma \in w_\sigma \llbracket Q'_\sigma \rrbracket$ in $\bar{T}_{\Sigma, \varphi}$
- (3) **THEN RETURN**(yes)
- (4) **ELSE** Construct $G_{\Sigma, \varphi}$ from Σ and φ ;
- (5) **IF** q_φ is reachable from q'_φ in $G_{\Sigma, \varphi}$
THEN RETURN(yes)
- (6) **ELSE RETURN**(no).

A comparison between Algorithm 3 and Algorithm 1 shows that adding structural keys to \mathcal{K} results in the additional steps between lines (1) to (3). These are justified by Lemma 3.

EXAMPLE 9. Let $\Sigma \cup \{\varphi\}$ be as in Example 8. Recall that φ is implied by Σ , and consider the tiny-tree $\bar{T}_{\Sigma, \varphi}$ in Figure 5. With this input instance Algorithm 3 returns yes in line (3) since there are two distinct author nodes in $\bar{T}_{\Sigma, \varphi}$ reachable from the publ node. \square

EXAMPLE 10. Let

$$\Sigma = \{(-^*, (\text{publ.author}, \{\text{first.S}, \text{last.S}\}))\}$$

and

$$\varphi = (-^*, (\text{publ}, \{\text{author.first.S}, \text{author.last.S}\})).$$

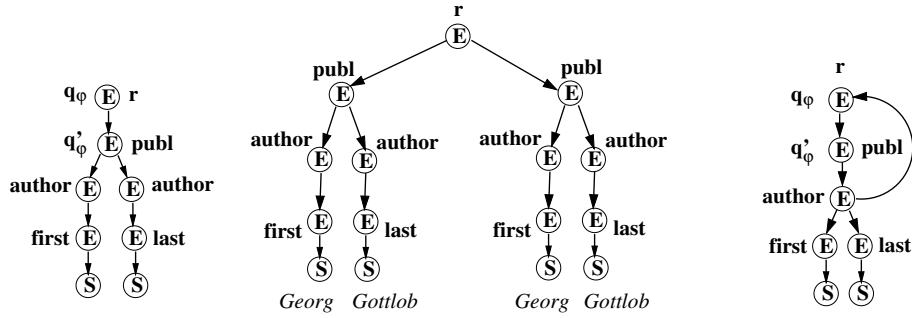


Figure 6: Witness graph, counterexample tree and another witness graph for Example 10.

The associated mini-tree $T_{\Sigma, \varphi}$ is the one illustrated in the very left of Figure 4. The tiny-tree $\bar{T}_{\Sigma, \varphi}$ coincides with this mini-tree since there are no structural keys in Σ . The witness graph $G_{\Sigma, \varphi}$ is illustrated in the left of Figure 6. Since q_φ is not reachable from q'_φ Algorithm 3 returns no. A counterexample tree for the implication of φ by Σ is illustrated in the middle of Figure 6. Let us now consider

$$\Sigma = \{(publ, (author, \emptyset)), (-^*, (publ.author, \{first.S, last.S\}))\}$$

and

$$\varphi = (-^*, (publ, \{author.first.S, author.last.S\})).$$

We know from previous examples that removing any single key from Σ means that φ is not implied by Σ . The mini- and tiny-trees are those of Figure 4, respectively. The witness graph $G_{\Sigma, \varphi}$ is illustrated in the right of Figure 6. Notice that

$$(-^*, (publ.author, \{first.S, last.S\}))$$

becomes applicable to φ in $\bar{T}_{\Sigma, \varphi}$. With this input instance Algorithm 3 returns yes in line (5) since q_φ is reachable from q'_φ . \square

The additional effort in lines (1)-(3) necessary for deciding key implication in \mathcal{K}' rather than \mathcal{K} does not add to the time-complexity, cf. Theorem 2.

THEOREM 3. *Let $\Sigma \cup \{\varphi\}$ be a finite set of XML keys in \mathcal{K}' . Algorithm 3 decides the implication problem $\Sigma \models \varphi$ in time $\mathcal{O}(|\varphi| \times (|\Sigma| + |\varphi|))$. \square*

7. BORDER TO INTRACTABILITY

The keys of the class \mathcal{K}' form a fragment of numerical constraints [20]. For the purposes of this paper it suffices to consider the following fragment of numerical constraints. An upper-bound constraint φ for XML is an expression

$$card(Q, (Q', S)) \leq \max$$

where Q, Q' are PL expressions, and all $P \in S$ are PL_s expressions such that $Q.Q'$ is a valid path expression if $k = 0$, and $Q.Q'.P$ are valid path expressions for all $P \in S$ if $k > 0$, where k is a non-negative integer, and where $\max \in \mathbb{N} \cup \{\infty\}$. An XML tree T satisfies $card(Q, (Q', \{P_1, \dots, P_k\})) \leq \max$ if and only if for all $q \in \llbracket Q \rrbracket$, for all $q' \in q \llbracket Q' \rrbracket$ such that for all x_1, \dots, x_k with $x_i \in q' \llbracket P_i \rrbracket$ for $i = 1, \dots, k$, the following holds:

$$|\{q'' \in q \llbracket Q' \rrbracket \mid \exists y_1, \dots, y_k \text{ such that } y_i \in q'' \llbracket P_i \rrbracket \text{ and } x_i =_v y_i \text{ for } i = 1, \dots, k\}| \leq \max.$$

However, it has been shown [20] that the implication problem of the class $\{card(\varepsilon, (P', S)) \leq \max \mid 1 \leq \max \leq 6\}$ is already *coNP*-hard to decide. Hence, our results show that \mathcal{K}' constitutes the large tractable fragment of upper-bound constraints where the upper bound is fixed to 1. A further tractable fragment of upper-bound constraints are numerical keys in which the set S of simple key path expressions is non-empty [20]. This fragment is incomparable to \mathcal{K}' .

8. CONCLUSION AND FUTURE WORK

We have demonstrated that the class \mathcal{K}' forms an expressive, yet tractable fragment of XML keys [8]. The class \mathcal{K}' extends the previously-studied robust class \mathcal{K} [21] by structural keys which identify nodes independently of any data values. The gain in expressiveness comes for free as our axiomatisation can be encoded in an algorithm that decides key implication in \mathcal{K}' as efficiently as for \mathcal{K} [21]. However, we could not characterise key implication in \mathcal{K}' as reachability in a suitable digraph, in contrast to \mathcal{K} . Moreover, \mathcal{K}' constitutes an efficiently-decidable subclass of upper-bound constraints for which implication is *strongly coNP*-hard to decide [20].

One area that warrants future research is the study of keys with respect to even more expressive path languages [6, 14, 27, 28, 38]. In particular, axiomatisability and implication problem are still open for XML keys with key path expressions in PL [9]. It may also prove a challenging task to apply the notion of value-equality, studied in this paper, to more expressive classes of XML constraints such as functional or multivalued dependencies [4, 19, 22, 25, 26, 30, 34, 35, 36].

9. ACKNOWLEDGEMENT

This research is supported by the Marsden fund council from Government funding, administered by the Royal Society of New Zealand.

10. REFERENCES

- [1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [2] V. Apparao et al. Document object model (DOM) Level 1 Specification, W3C Recommendation, Oct. 1998. <http://www.w3.org/TR/REC-DOM-Level-1/>.
- [3] M. Arenas, W. Fan, and L. Libkin. What's hard about XML schema constraints? In *Proceedings of the 13th International Conference Database and Expert Systems Applications - DEXA 2002*, number 2453 in

- Lecture Notes in Computer Science, pages 269–278. Springer, 2002.
- [4] M. Arenas and L. Libkin. A normal form for XML documents. *Trans. Database Syst.*, 29(1):195–232, 2004.
- [5] M. Arenas and L. Libkin. An information-theoretic approach to normal forms for relational and XML data. *J. ACM*, 52(2):246–283, 2005.
- [6] M. Benedikt, W. Fan, and G. Kuper. Structural properties of XPath fragments. *Theor. Comput. Sci.*, 336(1):3–31, 2005.
- [7] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0 (Fourth Edition) W3C Recommendation, Aug. 2006. <http://www.w3.org/TR/xml/>.
- [8] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. *Computer Networks*, 39(5):473–487, 2002.
- [9] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Reasoning about keys for XML. *Inf. Syst.*, 28(8):1037–1063, 2003.
- [10] P. Buneman, W. Fan, J. Siméon, and S. Weinstein. Constraints for semi-structured data and XML. *SIGMOD Record*, 30(1):47–54, 2001.
- [11] P. Buneman, W. Fan, and S. Weinstein. Path constraints in semistructured databases. *J. Comput. Syst. Sci.*, 61(2):146–193, 2000.
- [12] S. Davidson, W. Fan, and C. Hara. Propagating XML constraints to relations. *J. Comput. Syst. Sci.*, 73(3):316–361, 2007.
- [13] S. Davidson, W. Fan, and C. Hara. Erratum to “Propagating XML constraints to relations”. *J. Comput. Syst. Sci.*, 74(3):404–405, 2008.
- [14] A. Deutsch and V. Tannen. XML queries and constraints, containment and reformulation. *Theor. Comput. Sci.*, 336(1):57–87, 2005.
- [15] W. Fan. XML constraints. In *DEXA Workshops*, pages 805–809, 2005.
- [16] W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM*, 49(3):368–406, 2002.
- [17] W. Fan and J. Siméon. Integrity constraints for XML. *J. Comput. Syst. Sci.*, 66(1):254–291, 2003.
- [18] S. Hartmann, H. Köhler, S. Link, T. Trinh, and J. Wang. On the notion of an XML key. In *Proceedings of the 3rd International Workshop on Semantics in Data and Knowledge Bases - SDKB 2008*, number 4925 in Lecture Notes in Computer Science, pages 114–123. Springer, 2007.
- [19] S. Hartmann and S. Link. Characterising nested database dependencies by fragments of propositional logic. *Ann. Pure Appl. Logic*, 152(1-3):84–106, 2008.
- [20] S. Hartmann and S. Link. Numerical constraints for XML. In *Proceedings of the 14th International Workshop on Logic, Language, Information and Computation - WoLLIC 2007*, number 4576 in Lecture Notes in Computer Science, pages 203–217. Springer, 2007.
- [21] S. Hartmann and S. Link. Unlocking keys for XML trees. In *Proceedings of the 11th International Conference on Database Theory - ICDT 2007*, number 4353 in Lecture Notes in Computer Science, pages 104–118. Springer, 2007.
- [22] S. Hartmann and T. Trinh. Axiomatising functional dependencies for XML with frequencies. In *Proceedings of the 4th International Symposium on Foundations of Information and Knowledge Systems - FoIKS 2006*, number 3861 in Lecture Notes in Computer Science, pages 159–178. Springer, 2006.
- [23] D. Jungnickel. *Graphs, Networks and Algorithms*. Springer, 1999.
- [24] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0 W3C Recommendation Nov. 1999. <http://www.w3.org/TR/xpath>.
- [25] S. Link. On the Implication of Multivalued Dependencies in Partial Database Relations. *Int. J. Found. Comput. Sci.*, 19(3):691–715, 2008.
- [26] S. Link. Charting the completeness frontier of inference systems for multivalued dependencies. *Acta Inf.*, 45(7-8):565–591, 2008.
- [27] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004.
- [28] F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Logical Methods in Computer Science*, 2(3), 2006.
- [29] S. e. a. Pemberton. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition) W3C Recommendation, Jan. 2000. <http://www.w3.org/TR/xhtml1>.
- [30] L. V. Saxton and X. Tang. Tree Multivalued Dependencies for XML Datasets. In *Proceedings of the 5th International Conference on Advances in Web-Age Information Management - WAIM 2004*, number 3129 in Lecture Notes in Computer Science, pages 357–367. Springer, 2004.
- [31] D. Suciu. On database theory and XML. *SIGMOD Record*, 30(3):39–45, 2001.
- [32] H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures second edition, W3C Recommendation, Oct. 2004. <http://www.w3.org/TR/xmlschema-1/>.
- [33] V. Vianu. A web odyssey: from Codd to XML. *SIGMOD Record*, 32(2):68–77, 2003.
- [34] M. W. Vincent, J. Liu and C. Liu. A Redundancy Free 4NF for XML. In *Proceedings of the First International XML Database Symposium - XSym 2003*, number 2824 in Lecture Notes in Computer Science, pages 254–266. Springer, 2003.
- [35] M. W. Vincent, J. Liu and C. Liu. Strong functional dependencies and their application to normal forms in XML. *Trans. Database Syst.*, 29(3):445–462, 2004.
- [36] M. W. Vincent, J. Liu and M. K. Mohania. On the equivalence between FDs in XML and FDs in relations. *Acta Inf.*, 44(3-4): 207–247, 2007.
- [37] J. Wang. Using tree patterns for flexible handling of XML trees. Master’s thesis, *Massey University*, 2007.
- [38] P. Wood. Containment for XPath fragments under DTD constraints. In *Proceedings of the 9th International Conference on Database Theory - ICDT 2003*, pages 300–314, 2003.