

Promoting the Semantic Capability of XML Keys

Flavio Ferrarotti¹, Sven Hartmann², Sebastian Link¹, and Jing Wang³

¹ Victoria University of Wellington, New Zealand

² Clausthal University of Technology, Germany

³ Massey University, New Zealand

Flavio.Ferrarotti@vuw.ac.nz

Abstract. Keys for XML data trees can uniquely identify nodes based on the data values on some of their subnodes, either in the entire tree or relatively to some selected subtrees. Such keys have an impact on several XML applications. A challenge is to identify expressive classes of keys with good computational properties. In this paper, we propose such a new class of keys. In comparison to previous work, the new class of XML keys is defined using a more expressive navigational path language that allows the specification of single-label wildcards. This provides designers with an enhanced ability to capture properties of XML data that are significant for the application at hand. We establish a sound and complete set of inference rules that characterizes all keys that are implicit in the explicit specification of XML keys. Furthermore, we establish an efficient algorithm for deciding XML key implication.

1 Introduction

The study of integrity constraints has been recognized as one of the most important yet challenging areas of XML research [5,10]. The importance of XML constraints is due to a variety of applications ranging from schema design, query optimization, efficient storing and updating, data exchange and integration, to data cleaning [5]. However, for almost all classes of constraints the complex structure of XML data results in decision problems that are intractable. In particular, the consistency problem is either infeasible or intractable for several fragments of keys as defined by XML schema [2]. It is therefore a major challenge to find natural and useful classes of XML constraints that can be reasoned about efficiently [5,10]. Prime candidates of such classes are absolute and relative keys [3] that are defined on the basis of a tree model for XML as proposed by DOM [1] and XPath [4], but independently from schema specifications such as DTDs or XSDs [11]. Figures 1 and 2 show such a representation in which nodes are annotated by their type: *E* for element, *A* for attribute, and *S* for string (PCDATA).

Keys for XML can uniquely identify nodes based on the data values on some of their subnodes, either in the entire tree or relatively to some selected subtrees. In this context, keys are defined in terms of path expressions. Nodes are determined by (complex) values on some selected descendent nodes. In Figure 1, an example

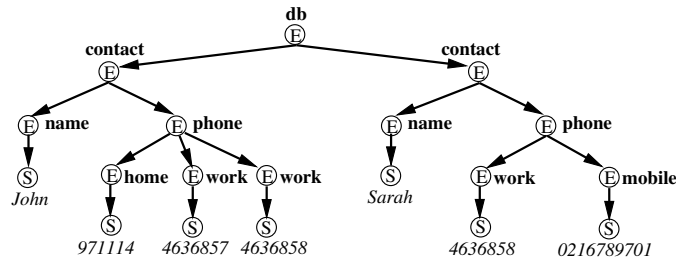


Fig. 1. Tree representation of XML data

of a reasonable absolute key is that a *contact* node can be uniquely identified in the whole document by values found on the leaves under its *name* and *phone* children. More precisely, assuming that we do not want two contacts with the same name and a phone number in common to appear in our XML agenda, our XML key should express that different *contact* nodes must differ on the values of their *name.S* descendent nodes, or otherwise their corresponding sets formed by the values of all their *phone...S* descendent nodes must be disjoint. Here the single-label wildcard “_” is a place holder for *work*, *home*, *mobile*, and any other child of a *phone* node. Note that the XML document in Figure 1 satisfies this key because even though both contacts share the phone number 4636858, they have different names.

In order to unlock the vast amount of application domains effectively it is crucial to define expressive notions of XML keys whose associated decision problems are still tractable. As shown in [7,8], for XML keys as defined in [3], the complexity of the implication problem is influenced by the choice of a path language for defining such XML keys. On the other hand, the choice of a navigational path language for defining XML keys influences the expressiveness of these keys. For instance, the class of XML keys with non-empty sets of simple key paths proposed in [3] is quite expressive. However, those XML keys cannot express the one from the previous example. A wider range of XML keys can be expressed by the class of XML keys proposed in [3] in which variable length wildcards in key paths as well as empty sets of key paths are allowed. But, up to our knowledge, the problems of axiomatising and developing efficient algorithms that decide the implication of such class of XML keys are still open (see [7,8]). By contrast, for the class of XML keys with non-empty sets of simple key paths, a sound and complete axiomatization and an efficient algorithm for the implication problem was presented in [7,8].

It is noteworthy that there are alternative XML representations of the example presented in Figure 1 in which the use of wildcards to specify the proposed XML key can be avoided. See for instance Figure 2. Nevertheless, the ability to interchange structure (the names) with data (their values) constitutes one of the strong points of semistructured data and XML. It allows users to generate intuitive XML documents, such as the one in Figure 1. In fact, it is unrealistic and counter-productive to generate less intuitive XML documents, for example the document in Figure 2, for the sake of making restricted tools applicable.

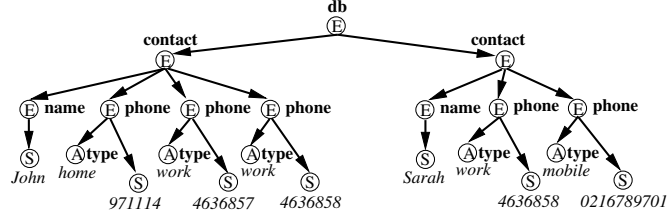


Fig. 2. Alternative tree representation of XML data in Figure 1

Thus, it is a significant challenge to identify expressive classes of XML keys that still enjoy good computational properties. In this paper, we propose such a new class of XML keys. In comparison to previous work, the new class utilizes a more expressive navigational path language that allows the specification of single-label wildcards. This provides designers with an enhanced ability to capture properties of XML data that are significant for the application at hand. In particular, the XML key $(\epsilon, \text{contact}, (\{\text{name.S}, \text{phone} \dots \text{S}\}))$ falls into this new class of keys, and captures successfully the semantics of our example. We establish a sound and complete set of inference rules that characterizes all keys that are implicit in the explicit specification of XML keys. Furthermore, we establish an efficient algorithm for deciding XML key implication.

2 Preliminaries

The XML Tree Model. 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. We refer to the elements of $\mathcal{L} = \mathbf{E} \cup \mathbf{A} \cup \{S\}$ as *labels*. An *XML tree* is a 6-tuple $T = (V, \text{lab}, \text{ele}, \text{att}, \text{val}, r)$ 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 $\text{lab}(v) \in \mathbf{E}$, an *attribute node* if $\text{lab}(v) \in \mathbf{A}$, and a *text node* if $\text{lab}(v) = S$. 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 $\text{ele}(v)$ is a list of element and text nodes in V and $\text{att}(v)$ is a set of attribute nodes in V . If v is an attribute or text node, then $\text{ele}(v)$ and $\text{att}(v)$ are undefined. The partial mapping val assigns a string to each attribute and text node: for each node $v \in V$, $\text{val}(v)$ is a string if v is an attribute or text node, while $\text{val}(v)$ is undefined otherwise. Finally, r is the unique and distinguished root node. 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 $i = 1, \dots, m$. The path p determines a word $\text{lab}(v_1) \dots \text{lab}(v_m)$ over the alphabet \mathcal{L} , denoted by $\text{lab}(p)$.

Value Equality of Nodes in XML Trees. Two nodes $u, v \in V$ are *value equal*, denoted by $u =_v v$, iff the following conditions are satisfied: (a) $\text{lab}(u) = \text{lab}(v)$, (b) if u, v are attribute or text nodes, then $\text{val}(u) = \text{val}(v)$, (c) if u, v are element nodes, then (i) if $\text{att}(u) = \{a_1, \dots, a_m\}$, then $\text{att}(v) = \{a'_1, \dots, a'_m\}$ 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 $\text{ele}(u) = [u_1, \dots, u_k]$, then $\text{ele}(v) = [v_1, \dots, v_k]$ and $u_i =_v v_i$ for $i = 1, \dots, k$.

Path Expressions for Node Selection in XML Trees. We use the path language $PL^{\{\cdot, \rightarrow, _ , _ * \}}$ consisting of expressions given by the following grammar: $Q \rightarrow \ell \mid \varepsilon \mid Q.Q \mid _ \mid _ *$. Here $\ell \in \mathcal{L}$ is any label, ε denotes the empty path expression, “ \cdot ” denotes the concatenation of two path expressions, “ $_$ ” denotes the *single-label* wildcard, and “ $_ *$ ” denotes the *variable length don't care* wildcard. Let P, Q be words from $PL^{\{\cdot, \rightarrow, _ , _ * \}}$. P is a *refinement* of Q , denoted by $P \lesssim Q$, if P is obtained from Q by replacing variable length wildcards in Q by words from $PL^{\{\cdot, \rightarrow, _ , _ * \}}$ and single-label wildcards in Q by labels from \mathcal{L} . Let Q be a word from $PL^{\{\cdot, \rightarrow, _ , _ * \}}$. 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 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, $v[[Q]] = \{w \mid T \models Q(v, w)\}$. We use $[[Q]]$ as an abbreviation for $r[[Q]]$ where r is the root node of T . For S a nonempty subset of $\{\cdot, \rightarrow, _ , _ *\}$, PL^S denotes the subset of $PL^{\{\cdot, \rightarrow, _ , _ * \}}$ expressions restricted to the constructs in S . $Q \in PL^{\{\cdot, \rightarrow, _ , _ * \}}$ is *valid* if it does not have labels $\ell \in \mathbf{A}$ or $\ell = S$ in a position other than the last one. Let P, Q be words from $PL^{\{\cdot, \rightarrow, _ , _ * \}}$. 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 $P \lesssim Q$ implies $P \subseteq Q$. For nodes v and v' of an XML tree T , the *value intersection* of $v[[Q]]$ and $v'[[Q]]$ is given by $v[[Q]] \cap_v v'[[Q]] = \{(w, w') \mid w \in v[[Q]], w' \in v'[[Q]], w =_v w'\}$.

3 Keys for XML

Following [3], let us first define formally the concept of XML key. Let PL_1, PL_2 and PL_3 be path languages suitable for node selection in XML trees such that for some fixed $1 \leq m \leq 3$, it holds that for $1 \leq i \leq 3$, if a path expression is in PL_i , then it is also in PL_m . An *XML key* φ in a class $\mathcal{K}(PL_1, PL_2, PL_3)$ of XML keys, is an expression of the form $(Q, (Q', \{Q_1, \dots, Q_k\}))$ where Q is a PL_1 expression, Q' is a PL_2 expression, and for $1 \leq i \leq k$, Q_i is a PL_3 expression such that $Q.Q'.Q_i$ is a valid PL_m expression. Herein, Q is called the *context path*, Q' is called the *target path*, and Q_1, \dots, Q_k are called the *key paths* of φ . An XML tree T satisfies a key $(Q, (Q', \{Q_1, \dots, Q_k\}))$ if and only if for every node $q \in [[Q]]$ and all nodes $q'_1, q'_2 \in q[[Q']]$ such that there are nodes $x_i \in q'_1[[Q_i]], y_i \in q'_2[[Q_i]]$ with $x_i =_v y_i$ for all $i = 1, \dots, k$, then $q'_1 = q'_2$. Some examples of XML keys follow.

- a. $(\varepsilon, (\text{contact}, \{\text{name}.S, \text{phone}._ .S\}))$. This XML key formalizes our example from the introduction in XML trees with the form of the tree in Figure 1.
- b. $(\varepsilon, (_ *. \text{contact}, \{\text{phone}._ .S\}))$. This key is not satisfied by the tree in Figure 1 since both contacts share the work phone number 4636858.
- c. $(\varepsilon, (\text{contact}, \{\text{name}.S, \text{phone}.S\}))$. This XML key expresses the same as the key in a., but over XML trees with the form of the tree in Figure 2.
- d. $(\varepsilon, (\text{contact}, \{\text{name}.S, \text{phone}._ *.S\}))$. Over trees with the form of the one in Figure 1, this XML key is equivalent to the key in a., and over trees with the form of the one in Figure 2, it is equivalent to the key in c.

- e. $(\varepsilon, (-^*.contact, \{-^*.S\}))$. It roughly expresses that any two different *contact* nodes in the tree must differ in all values of all their descendent *S* nodes. Neither the tree in Figure 1 nor the tree in Figure 2 satisfies this key.
- f. $(\varepsilon, (contact, \{\varepsilon\}))$. This key expresses that there cannot be two different *contact* nodes u and v such that both are children of the root node and the subtrees rooted at u and v are isomorphic by an isomorphism that is the identity on string values. It is satisfied by both the tree in Figure 1 and the tree in Figure 2.
- g. $(\varepsilon, (contact, \emptyset))$. This is satisfied by a tree if there is at most one *contact* node that is a child of the root node.
- h. $(\varepsilon, (-, \{-\}))$. In an XML tree that satisfies this key, one will never find two different children nodes of the root for which the value intersection of their corresponding sets of child nodes is non-empty. This key is satisfied by the tree in Figure 1 and violated by the tree in Figure 2.
- i. $(contact, (phone._, \{S\}))$. Interpreted in a XML tree with the form of the tree in Figure 1, this key expresses that, for each individual *contact* node, different *work*, *home* and *mobile* nodes must all have different phone numbers.

Let $\Sigma \cup \{\varphi\}$ be a finite set of XML keys in a class \mathcal{C} . We say that Σ (*finitely*) *implies* φ , denoted by $\Sigma \models_{(f)} \varphi$, if and only if every (finite) XML tree T that satisfies all $\sigma \in \Sigma$ also satisfies φ . The (*finite*) *implication problem* for \mathcal{C} is to decide, given any finite set $\Sigma \cup \{\varphi\}$ of keys in \mathcal{C} , whether $\Sigma \models_{(f)} \varphi$. For a set Σ of keys in \mathcal{C} , let $\Sigma_{(f)}^* = \{\varphi \in \mathcal{C} \mid \Sigma \models_{(f)} \varphi\}$ be its (*finite*) *semantic closure*, i.e., the set of all keys (finitely) implied by Σ . The notion of syntactical inference ($\vdash_{\mathfrak{R}}$) with respect to a set \mathfrak{R} of inference rules can be defined analogously to the notion in the relational data model. That is, a finite sequence $\gamma = [\gamma_1, \dots, \gamma_l]$ of XML keys is called an *inference from Σ by \mathfrak{R}* if every γ_i is either an element of Σ or is obtained by applying one of the rules of \mathfrak{R} to appropriate elements of $\{\gamma_1, \dots, \gamma_{i-1}\}$. We say that the inference γ *infers* γ_l , i.e. the last element of the sequence γ , and write $\Sigma \vdash_{\mathfrak{R}} \gamma_l$. For a finite set Σ of keys in \mathcal{C} , let $\Sigma_{\mathfrak{R}}^+ = \{\varphi \mid \Sigma \vdash_{\mathfrak{R}} \varphi\}$ be its *syntactic closure* under inferences by \mathfrak{R} . A set \mathfrak{R} of inference rules is said to be *sound (complete)* for the (finite) implication of keys in \mathcal{C} if for every finite set Σ of XML keys in \mathcal{C} we have $\Sigma_{\mathfrak{R}}^+ \subseteq \Sigma_{(f)}^*$ ($\Sigma_{(f)}^* \subseteq \Sigma_{\mathfrak{R}}^+$). The set \mathfrak{R} is said to be an *axiomatisation* for the (finite) implication of keys in \mathcal{C} if \mathfrak{R} is both sound and complete for the (finite) implication of keys in \mathcal{C} . Finally, an axiomatisation \mathfrak{R} is said to be *finite* if the set \mathfrak{R} is finite.

A finite axiomatization for the (finite) implication of keys in the class of XML keys with non-empty sets of simple key paths $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_+^{\{\dots\}})$, was established in [7,8]. Here we use the “+” symbol to indicate that the finite set of key paths must not be empty. It is important to note that from a practical point of view, and as illustrated by the examples given in [3,7,8], many useful XML keys belong to the class $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_+^{\{\dots\}})$. Note that the XML keys in points (c) and (f) above belong to this class.

A more expressive class of XML keys was studied in [9], namely the class $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_*^{\{\dots\}})$. Here we use the “*” symbol to stress that the finite set of key paths can be empty. Note that, the key in point (g) above belongs

to this class of XML keys. Also the keys in points (c) and (f) belong to this class since those keys also belong to the class $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_+^{\{\dots\}})$ which is strictly included in it.

In this paper, we study the class $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_+^{\{\dots\}})$ of XML keys in which single-label wildcards are allowed in context, target and key paths. Except for the keys in points (d), (e) and (g), all other keys introduced above belong to this class. Clearly, this class of XML keys is more expressive than the class studied in [7,8] as it strictly includes all XML keys that belong to $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_+^{\{\dots\}})$ and, at the same time, it includes keys as those in points (a), (b), (h) and (i) for which there are no equivalent XML keys in $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_+^{\{\dots\}})$. Note that this new class is orthogonal to the class $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_*^{\{\dots\}})$ studied in [9]. Take for instance the key in point (h), it is easy to see that there is no XML key in $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_*^{\{\dots\}})$ that is satisfied by exactly those XML trees that satisfy $(\varepsilon, (-, \{-\}))$. A clear example in the other direction is given by the XML key in point (g). Also note that the set of inference rules used in [9] for the finite axiomatization of the class $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_*^{\{\dots\}})$ is considerably different from the set of inference rules proposed in this work for the class $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_+^{\{\dots\}})$. Furthermore, the techniques and results used in that paper are different.

From now on, for an XML key φ , we use Q_φ to denote its context path, Q'_φ to denote its target path, and $P_1^\varphi, \dots, P_{k_\varphi}^\varphi$ to denote its key paths, where k_φ is the number of its key paths.

4 An Axiomatisation

Theorem 1. *Let \mathfrak{R} denote the set of inference rules in Table 1 together with the key-path containment rule in Equation (1). Then \mathfrak{R} forms a finite axiomatisation for the implication of XML keys in $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_+^{\{\dots\}})$.*

$$\frac{(Q, (Q', \mathcal{S} \cup \{P\}))}{(Q, (Q', \mathcal{S} \cup \{P'\}))} P' \subseteq P \quad (\text{key-path containment}) \quad (1)$$

The soundness of the *key-path containment rule* is not difficult to see. Suppose an XML tree T violates $(Q, (Q', \mathcal{S} \cup \{P'\}))$. Then there is some node $q \in \llbracket Q \rrbracket$ and some nodes $q'_1, q'_2 \in q \llbracket Q' \rrbracket$ such that $q'_1 \neq q'_2$ and for all $P_i \in \mathcal{S} \cup \{P'\}$ it holds that $q'_1 \llbracket P_i \rrbracket \cap_v q'_2 \llbracket P_i \rrbracket \neq \emptyset$. In particular, this means that $q'_1 \llbracket P' \rrbracket \cap_v q'_2 \llbracket P' \rrbracket \neq \emptyset$. Since $P' \subseteq P$, we have that $q'_1 \llbracket P' \rrbracket \subseteq q'_1 \llbracket P \rrbracket$ and $q'_2 \llbracket P' \rrbracket \subseteq q'_2 \llbracket P \rrbracket$. But then, $q'_1 \llbracket P \rrbracket \cap_v q'_2 \llbracket P \rrbracket \neq \emptyset$. Hence, T also violates $(Q, (Q', \mathcal{S} \cup \{P\}))$. It is a simple technical task to modify the soundness proofs of the rules in Table 1, to show that they remain sound.

For the completeness proof we need to show that, for an arbitrary finite set $\Sigma \cup \{\varphi\}$ of keys in $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_+^{\{\dots\}})$, if $\varphi \notin \Sigma_{\mathfrak{R}}^+$, then there is some XML tree T which is a counter-example for the implication of φ by Σ .

The general proof strategy is as follows. In a first step, we represent φ in terms of a finite node-labelled tree $T_{\Sigma, \varphi}$, which we call the *mini-tree*. Then, we calculate the impact of each key in Σ on the counter-example tree T that we

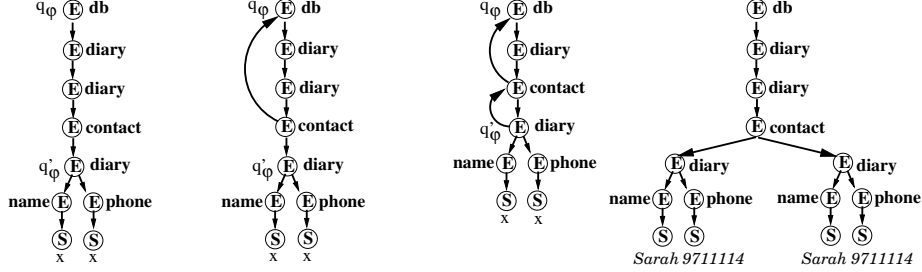


Fig. 3. A mini-tree, two witness graphs and a counter-example tree

impact from those that do not, we introduce the notion of *applicability*. Let $T_{\Sigma, \varphi}$ be the mini-tree of the key φ with respect to Σ , and let \mathcal{M} be its marking. 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 . As an example, consider Σ and φ from Example 1. While σ_1 is applicable to φ , σ_2 is not. 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 , $G_{\Sigma, \varphi}$ contains the directed edge (w'_σ, w_σ) from w'_σ to w_σ .

Example 2. Let Σ and φ be as in Example 1. The witness graph $G_{\Sigma, \varphi}$ is shown as second picture in Figure 3. It contains a witness edge arising from σ_1 .

Theorem 2. Let $\Sigma \cup \{\varphi\}$ be a finite set of keys in $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL^{\{\dots\}}_+)$. We have $\Sigma \models \varphi$ if and only if q_φ is reachable from q'_φ in $G_{\Sigma, \varphi}$.

The next example illustrates how the edges of the witness graph encode an inference by \mathfrak{R} .

Example 3. Let Σ consists of the two keys $\sigma_1 = (\varepsilon, (\text{diary.contact}, \{_S, _S\}))$ and $\sigma_2 = (\text{diary}, (\text{contact}_., \{_ \}))$, and let $\varphi = (\varepsilon, (\text{diary.contact}_., \{_S, _S\}))$. The witness graph $G_{\Sigma, \varphi}$ is shown as first picture of Figure 4. Here *detail* is the fixed label chosen from $\mathbf{E} - \mathcal{L}_{\Sigma, \varphi}$. We use the same fixed label for all cases that follow. We apply the *subnodes* rule to σ_2 to derive $\sigma'_2 = (\text{diary}, (\text{contact}_., \{_ \}))$, the new *key-path containment* rule to σ'_2 to derive $\sigma''_2 = (\text{diary}, (\text{contact}_., \{_S\}))$ and the *superkey* rule to σ''_2 to obtain $\sigma'''_2 = (\text{diary}, (\text{contact}_., \{_S, _S\}))$. Let $\Sigma' = \{\sigma_1, \sigma'''_2\}$. The witness graph $G_{\Sigma', \varphi}$ is shown as second picture in Figure 4. We apply *context-target* to σ'''_2 to derive $(\text{diary.contact}, (_., \{_S, _S\}))$ denoted by σ_3 . Let $\Sigma'' = \{\sigma_1, \sigma_3\}$. The corresponding witness graph $G_{\Sigma'', \varphi}$ is shown as third picture in Figure 4. An application of the *interaction* rule to σ_1 and σ_3 results in φ which shows that $\varphi \in \Sigma''_{\mathfrak{R}}$, illustrated on the right of Figure 4.

Note that, if we simply apply the construction of the mini-trees from [7,8], i.e., if we simply replace each variable length wildcard “ $_*$ ” by the single label ℓ_0 instead of a sequence of $l + 1$ labels ℓ_0 , then Theorem 2 does not hold. In fact,

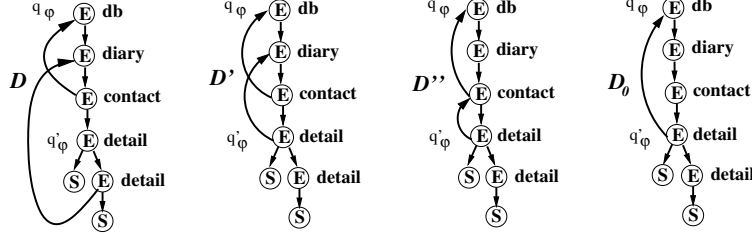


Fig. 4. Encoding the keys from Example 3 as witness edges

for the mini-tree of φ in Example 1, we would get the witness graph shown as third picture in Figure 3. Hence, q_φ would be reachable from q'_φ but $\varphi \notin \Sigma^*$ as the tree on the right of Figure 3 shows.

An Illustration of The Completeness Argument. Let $\Sigma \cup \{\varphi\}$ be an arbitrary finite set of keys in $\mathcal{K}(PL^{\{\cdot, \rightarrow, *\}}, PL^{\{\cdot, \rightarrow, *\}}, PL^{\{\cdot, \rightarrow\}})$ such that $\varphi \notin \Sigma_{\mathfrak{R}}^+$. In order to show completeness one needs to demonstrate that $\varphi \notin \Sigma^*$. Indeed, we can construct a finite XML tree T which satisfies all keys in Σ but does not satisfy φ . If $\varphi \notin \Sigma_{\mathfrak{R}}^+$, then one can show that there is no path from q'_φ to q_φ in $G_{\Sigma, \varphi}$. Let u denote the bottom-most descendant node of q_φ in $T_{\Sigma, \varphi}$ such that q_φ is still reachable from u in $G_{\Sigma, \varphi}$. Consequently, u is a proper ancestor of q'_φ because otherwise u and thus q_φ were reachable from q'_φ in $G_{\Sigma, \varphi}$ according to the downward edges. Let T_0 denote a copy of the path from r to u , and T_1, T_2 denote two node-disjoint copies of the subtree of $T_{\Sigma, \varphi}$ rooted at u . We want that a node of T_1 and a node of T_2 become value equal precisely when they are copies of the same marked node in $T_{\Sigma, \varphi}$. For attribute and text nodes this is achieved by choosing string values accordingly, while for element nodes we can adjoin a new child node with a label from $\mathcal{L} - (\mathcal{L}_{\Sigma, \varphi} \cup \{\ell_0\})$ to achieve this. The counter-example tree T is obtained from T_0, T_1, T_2 by identifying the leaf node u of T_0 with the root nodes of T_1 and T_2 .

Example 4. Let Σ and φ be as in Example 2. Then u is the single *contact*-node in $T_{\Sigma, \varphi}$. The counter-example tree T , shown on the right of Figure 3, is the result of this construction.

5 Deciding Implication in $\mathcal{K}(PL^{\{\cdot, \rightarrow, *\}}, PL^{\{\cdot, \rightarrow, *\}}, PL^{\{\cdot, \rightarrow\}})$

Theorem 2 suggests to utilise the following algorithm for deciding XML key implication.

Algorithm 3 (XML Key Implication in $\mathcal{K}(PL^{\{\cdot, \rightarrow, *\}}, PL^{\{\cdot, \rightarrow, *\}}, PL^{\{\cdot, \rightarrow\}})$)

Input: finite set $\Sigma \cup \{\varphi\}$ of XML keys in $\mathcal{K}(PL^{\{\cdot, \rightarrow, *\}}, PL^{\{\cdot, \rightarrow, *\}}, PL^{\{\cdot, \rightarrow\}})$

Output: yes, if $\Sigma \models \varphi$; no, otherwise

Method:

(1) Construct $G_{\Sigma, \varphi}$ for Σ and φ ;

(2) if q_φ is reachable from q'_φ in $G_{\Sigma, \varphi}$ then return(yes); else return(no).

For the class $\mathcal{K}(PL^{\{\cdot, \rightarrow, *\}}, PL^{\{\cdot, \rightarrow, *\}}, PL^{\{\cdot, \rightarrow\}})$, Algorithm 3 can be implemented in time quadratic in the size of the input [7,8]. Due to the different construction

needed for the mini-trees, it is not clear whether Algorithm 3 can also be implemented in quadratic time for the class $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_+^{\{\dots\}})$. However, it can still be implemented efficiently.

Theorem 4. *If $\Sigma \cup \{\varphi\}$ is a finite set of keys in $\mathcal{K}(PL^{\{\dots\}}, PL^{\{\dots\}}, PL_+^{\{\dots\}})$, then the implication problem $\Sigma \models \varphi$ can be decided in $\mathcal{O}(|\varphi| \times l \times (|\Sigma| + |\varphi| \times l))$ time, where $|\varphi|$ is the sum of the lengths of all path expressions in φ , $|\Sigma|$ is the sum of all sizes $|\sigma|$ for $\sigma \in \Sigma$, and l is the maximum number of consecutive single-label wildcards that occur in Σ .*

6 Future Work

When defining XML keys there are at least two factors that determine their expressiveness: i) the navigational operators for accessing nodes (e.g., path expressions for descendants/ancestors) and ii) the notion of value equal nodes (e.g., string equality on leaves, isomorphic subtrees). It is a challenging goal to identify (combinations of) such sources of expressiveness that still permit the development of tractable solutions to the associated decision problems [6].

Future research should further address the implementation of reasoning techniques in real XML applications, e.g., in schema design, data integration, exchange and cleaning, as well as query optimisation and rewriting, and consistent query answering. This may lead to the detection of further fragments of XML keys or relaxations thereof that deserve investigation.

References

1. Apparao, V., et al.: Document object model (DOM) level 1 specification, W3C recommendation (1998), <http://www.w3.org/TR/REC-DOM-Level-1/>
2. Arenas, M., Fan, W., Libkin, L.: On the complexity of verifying consistency of XML specifications. *SIAM J. Comput.* 38(3), 841–880 (2008)
3. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Keys for XML. *Computer Networks* 39(5), 473–487 (2002)
4. Clark, J., DeRose, S.: XML path language (XPath) version 1.0, W3C recommendation (1999), <http://www.w3.org/TR/xpath>
5. Fan, W.: XML constraints. In: DEXA Workshops, pp. 805–809. IEEE Computer Society, Los Alamitos (2005)
6. Hartmann, S., Koehler, H., Link, S., Trinh, T., Wang, J.: On the notion of an XML key. In: Schewe, K.-D., Thalheim, B. (eds.) SDKB 2008. LNCS, vol. 4925, pp. 103–112. Springer, Heidelberg (2008)
7. Hartmann, S., Link, S.: Unlocking keys for XML trees. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 104–118. Springer, Heidelberg (2006)
8. Hartmann, S., Link, S.: Efficient reasoning about a robust XML key fragment. *ACM Trans. Database Syst.* 34(2) (2009)
9. Hartmann, S., Link, S.: Expressive, yet tractable XML keys. In: EDBT. ACM Int. Conf. Proceeding Series, vol. 360, pp. 357–367. ACM, New York (2009)
10. Suciu, D.: On database theory and XML. *SIGMOD Record* 30(3), 39–45 (2001)
11. Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N.: XML Schema Part 1: Structures, 2nd edn., W3C Recomm (2004), <http://www.w3.org/TR/xmlschema-1/>