Erratum: A Correction to "Propagating XML Constraints to Relations"

Susan Davidson, Wenfei Fan, Carmem Hara

Abstract

This erratum reports a bug in [2], and provides a fix without negative impact on the main results of [2] (the propagation algorithms and their complexity bounds).

1. The Bug

Background. The objective of [2] is to provide a technique for normalizing the design of relational storage of XML data. For XML shredding, i.e., transformations from XML data to relations, it aims to derive a normalized relational schema, in BCNF or 3NF, for storing XML data. The idea is based on XML key propagation: given a set Σ of XML keys and an XML shredding mapping (transformation) σ from XML data to relations of a schema $\mathcal{R}$, it first computes a minimum cover F of relational functional dependencies (FDs) propagated from Σ, and then normalizes $\mathcal{R}$ by leveraging the FDs of F. Here an FD $\psi \in F$ is *propagated* from Σ, denoted by $\Sigma \models_\sigma \psi$, if for any XML tree T that satisfies Σ, its relational encoding $\sigma(T)$ satisfies ψ.

In addition to proposing this normalized XML shredding approach, [2] shows that the propagation analysis is undecidable for XML constraints including both keys and foreign keys, and for XML shredding that is "relationally complete". It then focuses on a class of XML shredding mappings and a subclass $\mathcal{K}_{att}$ of XML keys studied in [1]. In this setting, it provides a PTIME algorithm for computing a minimum cover of relational FDs propagated from XML keys.

XML keys of $\mathcal{K}_{att}$ are of the form $\varphi = (Q, (Q', S))$, where Q, Q' are XPath expressions defined in terms of the child and descendant-or-self axes, and S is a set of XML attributes that are required to exist. In an XML tree T, let $n[\![Q]\!]$ denote the set of nodes reached via an XPath expression Q from a node n; in particular, when n is the root r of T, we simply write $n[\![Q]\!]$ as $[\![Q]\!]$. Then T satisfies φ, denoted by $T \models \varphi$, if for any node $v \in [\![Q]\!]$, (Q', S) is a key for the subtree rooted at v, i.e., for any two nodes u_1, u_2 in $v[\![Q']\!]$, if u_1 and u_2 agree on their attributes in S (for any $@l \in S$, $u_1.@l = u_2.@l$), then u_1 and u_2 must be the same node.

An XML shredding mapping of [2] is defined in terms of XPath with the child and descendant-or-self axes, and a list of variables $[x_1, \ldots, x_m]$. It consists of rules of the form: $\texttt{Rule}(R) = \{l_1 : \mathsf{value}(x_1), \ldots, l_k : \mathsf{value}(x_k)\}$, where $x_i \leftarrow x_j/P_i$, $j < i$, $x_1 = r$, and P_i is an XPath expression with child axes, and with descendant-or-self axes if $i = 1$. Given an XML tree T, it populates an instance of R by extracting data from T via P_i's, and assigning the data as the value of field l_i, top-down from the root of T.

The propagation algorithm of [2] makes use of the *implication analysis of XML keys*, to determine whether a set Σ of XML keys entails another XML key φ, denoted by $\Sigma \models \varphi$. To this end [2] capitalizes on the inference system of XML keys developed in [1], tailored for the subclass $\mathcal{K}_{att}$ of keys.

The Mistake. As pointed out by [4], the inference system proposed for XML keys of [1] is neither sound nor complete. The counterexample given in [4] consists of XML keys of a "special" form $(Q, (Q', S))$, where S contains the empty path ϵ. As a specific form of the more general system of [1], the inference system $\mathcal{I}_{att}$ for the subclass $\mathcal{K}_{att}$ is also *incomplete*, although it is *sound*. That is, for any set Σ of XML keys and a single key φ, if φ can be proved from Σ via $\mathcal{I}_{att}$ rules, then $\Sigma \models \varphi$; however, when φ is of a *special form* $(\epsilon, (Q', \emptyset))$, it is possible that φ cannot be deduced from Σ via $\mathcal{I}_{att}$ although $\Sigma \models \varphi$, as illustrated by the example below.

Example 1.1: Consider the set of XML keys $\Sigma = \{(a, (//b, \emptyset))\}$, and a single XML key $\varphi = (\epsilon, (a/b/b, \emptyset))$. Then for any XML tree T that satisfies Σ, T also satisfies φ. The reason is simple: Σ assures that there exists at most one b-node in any subtree rooted at an a-node in T. As a result T trivially satisfies φ since there exist no nodes in T reached by following path $a/b/b$, i.e., $|[\![a/b/b]\!]| = 0$. However, φ cannot be proved from Σ using $\mathcal{I}_{att}$.

Now consider a transformation $\sigma = (\texttt{Rule}(R))$, where $\texttt{Rule}(R) = \{\bar{l}\colon \mathsf{value}(x)\}$, and $x \leftarrow r/a/b/b$. Given an XML tree T that satisfies Σ, the relation $\sigma(T)$ shredded from T satisfies the FD $\emptyset \rightarrow l$. Indeed, from the discussion above it follows that there exists no node in T reached by following path $a/b/b$, which is used to populate field l in R. Thus, l is given the special constant `null`. Since $\mathcal{I}_{att}$ fails to derive φ from Σ, the propagation algorithms also fail to conclude that $\emptyset \rightarrow l$ is propagated from Σ via σ. □

The Impact. As will be formally shown in Section 2, for any set Σ of XML keys, the only XML keys that may not be derived from Σ using $\mathcal{I}_{att}$ are of the form $(\epsilon, (Q', \emptyset))$ such that for *any* XML tree T, if $T \models \Sigma$, then *no nodes* can be reached from the root r of T via path Q', i.e., $|[\![Q']\!]| = 0$. The keys of the form $(\epsilon, (Q', S))$ are referred to as *absolute keys* [1, 2]. We say that Σ *prevents path* Q' if $|[\![Q']\!]| = 0$ in any XML tree that satisfies Σ. Then keys that $\mathcal{I}_{att}$ fails to derive are absolute keys defined with a path that is prevented by Σ. Such keys are "nonexistence" constraints as they concern paths that *cannot exist*. In contrast, the XML keys that [1, 2] focus on are "value-based", which are what people commonly use in practice.

As a result, FDs that the propagation algorithms of [2] fail to detect, due to the incompleteness of $\mathcal{I}_{att}$, are of the form $\emptyset \rightarrow l$ (and others that can be derived from them using Armstrong Axioms), where l is a field in a relation for storing XML data, such that the value of the l field is *guaranteed to be* the special constant `null` in the entire relation.

It should be remarked that FDs of this form are *not needed* in the normalization of relations for storing XML data, which is the main focus of [2].

A Fix to the bug. Since such FDs have no impact on the normalization analysis of relations for XML shredding,

$$\frac{(Q,\,(Q',\emptyset)),\quad Q_1 \subseteq Q',\quad Q_1/Q_2 \subseteq Q',\quad \epsilon \notin Q_2}{(\epsilon,\,(Q/Q_1/Q_2//,S)),\quad \text{where } S \text{ is any set of attributes}} \qquad \text{(double-path)}$$

$$\frac{\begin{array}{c}(Q,(Q'/@l,\emptyset)),Q_1 \subseteq Q',\quad Q_1/Q_2 \subseteq Q',\\ (Q/Q_1,\,(\epsilon,\,\{@l\})),\quad (Q/Q_1/Q_2,(\epsilon,\{@l\})),\quad \epsilon \notin Q_2\end{array}}{(\epsilon,\,(Q/Q_1/Q_2//,S)),\quad \text{where } S \text{ is any set of attributes}} \qquad \text{(double-attribute)}$$

$$\frac{\begin{array}{c}(Q,(Q'/@l,\emptyset)),\quad Q_1 \subseteq Q',\\ Q_1/Q_2 \subseteq Q',\quad (Q/Q_1,\,(\epsilon,\{@l\})),\quad \epsilon \notin Q_2\end{array}}{(\epsilon,\,(Q/Q_1/Q_2/@l,\emptyset))} \qquad \text{(attribute-on-a-path)}$$

Figure 1: Additional Inference rules for XML key implication

one may simply ignore FDs of this form and nonexistence keys. Nevertheless, in the next section we present additional rules for inference analysis of XML keys of [2], such that the extended system is *sound and complete* for XML keys of $\mathcal{K}_{att}$ including nonexistence keys. We also provide an algorithm for determining implication of XML keys of $\mathcal{K}_{att}$, and thus shows that the implication problem remains to be in PTIME. In fact, the complexity bound for the implication analysis is precisely *the same as* its counterpart given in [2].

Based on this, the algorithms for the propagation analysis of [2] can be simply modified to adopt a preprocessing phase as follows. For each $\texttt{Rule}(R) = \{l_1 : \mathsf{value}(x_1), \ldots, l_k : \mathsf{value}(x_k)\}$, and each $i \in [1,k]$, first find the path expression ρ_i from the root to x_i, and then test whether the given set Σ of XML keys *prevents* ρ_i. If so, then l_i contains the special constant `null` in the entire relation, and thus it is useless and ignored. This can be done in PTIME, by invoking function `prevent` of the algorithm for XML key implication analysis, given in Figure 4. After this preprocessing, the propagation algorithms of [2] can be used to derive FDs propagated from Σ, without considering those useless l_i fields, in PTIME as shown in [2].

2. Corrections to XML Key Inference

Below we first provide additional inference rules to make $\mathcal{I}_{att}$ sound and complete for XML keys $\mathcal{K}_{att}$ of [2], including those nonexistence keys. We then present a modification of the algorithm of [2] for determining XML key implication, and show that the implication analysis is in PTIME.

To make this erratum self-contained, we include part of the proofs and algorithms already presented in [2].

2.1 Axiomatization

We give three inference rules, shown in Figure 1, for deriving nonexistence keys. We refer to the set consisting of these three rules and the inference rules given in [2] also as $\mathcal{I}_{att}$.

The new rules are illustrated as follows.

- *double-path:* given any node n in $[\![Q]\!]$, if there exists a single node reached by following Q' from n, and n_1 is in $n[\![Q_1]\!]$ for some $Q_1 \subseteq Q'$, then there can exist no node n_2 in $n[\![Q_1/Q_2]\!]$ if $Q_1/Q_2 \subseteq Q'$. Otherwise, n_2 would also be an element of $n[\![Q']\!]$. Since such a node does not exist in any subtree rooted at a node in $[\![Q]\!]$, there exist *no* nodes reached by following path $Q/Q_1/Q_2$ from the root. Thus, any path following Q_2 is also guaranteed not to exist, and any set of attributes is a key for these nodes.
- *double-attribute:* given any node n in $[\![Q]\!]$, if every node n_1 in $n[\![Q_1]\!]$ is required to have an attribute $@l$, every node n_2 in $n[\![Q_1/Q_2]\!]$ is also required to have an attribute $@l$, but in the subtree rooted at n there can exist at most one $@l$ attribute, then such n_2 *does not* exist in the tree. Since n_2 is guaranteed not to exist, any path following Q_2 is also guaranteed not to exist, and they can be keyed by any set of attributes S.
- *attribute-on-a-path:* given any node n in $[\![Q]\!]$ and n_1 in $n[\![Q_1]\!]$, if n_1 is required to have an attribute $@l$, but in the subtree rooted at n there can exist at most one $@l$ attribute, then there can exist no nodes reached via path $Q/Q_1/Q_2/@l$, when Q_2 is not ϵ or "//".

As remarked earlier, these additional inference rules are for deriving nonexistence keys, i.e., keys defined with paths that are prevented by a given set of XML keys. As nonexistence of a path P is not expressible in the keys languages of [1, 2], we can only "approximate" it by $|[\![P]\!]| \leq 1$.

Theorem 2.1: *The* extended *inference system $\mathcal{I}_{att}$ is sound and complete for determining implication of $\mathcal{K}_{att}$ keys.* □

Proof: Consider any set $\Sigma \cup \{\varphi\}$ of keys in $\mathcal{K}_{att}$, where $\varphi = (Q,(Q',\{@A_1,\ldots,@A_k\}))$. The soundness of $\mathcal{I}_{att}$ can be verified by induction on the lengths of $\mathcal{I}_{att}$-proofs. The proof of completeness relies on the notion of abstract trees. An *abstract tree* is an extension of an XML tree by allowing "//" as a node label, which is treated as an ordinary tag. Observe that in an abstract tree T, the sequence of labels on a path is a path expression that possibly contains occurrences of "//". Let ρ be the sequence of labels in the path from node a to b in T, and P be any path expression. We say that $T \models P(a,b)$ if $\rho \subseteq P$. Given this, the definitions of node sets and satisfaction of constraints in $\mathcal{K}_{att}$ can be easily generalized for abstract trees. Abstract trees have the following property (see [1] for a proof):

Lemma 2.2: *If there is an abstract tree T such that $T \models \Sigma$ and $T \models \neg\varphi$, then there is an XML tree G such that $G \models \Sigma$ and $G \models \neg\varphi$.* □

Given the notion of abstract trees, we have to show that if $\Sigma \models \varphi$ then $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$. The completeness proof consists of two parts. The first part focuses on deriving nonexistence keys. Note that if Σ prevents path Q/Q' (where $\varphi = (Q,(Q',S))$ then φ is trivially satisfied and $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$. The second part shows the completeness of $\mathcal{I}_{att}$ when φ is "value-based", i.e., when Σ does not prevent path Q/Q'. That is, if $\Sigma \models \varphi$ based on their key values then $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$.

Part 1. We show the first part of the proof as follows. Suppose that $\Sigma \not\vdash_{\mathcal{I}_{att}} \varphi$. Then we show that Σ does not prevent path Q/Q' by constructing an XML tree G such that $G \models \Sigma$, and $[\![Q/Q']\!] > 0$. In other words, if Σ prevents Q/Q' then $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$.

The construction of G involves the following steps: First, we define an abstract tree T such that T contains a single path Q/Q'. Then, T is modified in a way that the resulting tree T_f satisfies Σ. That is, for each $\phi \in \Sigma$, we check whether T satisfies ϕ. If not, certain nodes in T are removed such that the modified tree satisfies ϕ. At the end of the process, $T_f \models \Sigma$ and in T_f either: (1) $|[\![Q/Q']\!]| = 1$, and we construct an XML tree G from T_f that satisfies Σ and $|[\![Q/Q']\!]| = 1$; or (2) $|[\![Q/Q']\!]| = 0$. In this case, we show that our assumption that $\Sigma \not\vdash_{\mathcal{I}_{att}} \varphi$ does not hold. That is, we show that each removal operation corresponds to the application of certain rules in $\mathcal{I}_{att}$. Therefore, if $|[\![Q/Q']\!]| = 0$ then $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$, which contradicts the assumption.

We start by giving the construction of an abstract tree T. As illustrated in Figure 2(a), T consists of a single path Q/Q' from the root leading to a node n. For each element v in T we add all the attributes required by Σ. That is, v has attributes $\{@a \mid v \in [\![Q]\!],\ (Q_1, (Q_1', S)) \in \Sigma,\ Q \subseteq Q_1/Q_1',\ @a \in S\}$. A distinct value is assigned to each attribute created.

We next modify the abstract tree T such that $T \models \Sigma$. We examine each ϕ in Σ, and if T does not satisfy ϕ, we remove certain nodes from T such that the modified tree satisfies ϕ.

repeat until no further change in T
  if there exist key $\phi = (Q_\phi, (Q'_\phi, \emptyset)) \in \Sigma$, and nodes w, x, y in T
    such that $T \models Q_\phi(r, w) \wedge Q'_\phi(w, x) \wedge Q'_\phi(w, y) \wedge x \neq y$,
      where x is closer to the root than y
  then
    if $Q'_\phi = P/@l$ and $@l$ is enforced by Σ to exist for parent(y)
      then remove subtree rooted at the parent of y
    else remove subtree rooted at y

Observe that keys in Σ with a non-empty set of key paths may not cause any removals since in the construction of T all attributes have been given distinct values. Thus, the process only considers keys with an empty set of key paths.

The process terminates since T is finite and thus removals can be performed only finitely many times. Note that T_f, the tree obtained upon termination of the process, satisfies Σ. If $[\![Q/Q']\!] = 1$ in T_f, then the process did not remove any nodes, and we can construct an XML tree G from T_f by replacing each occurrence of // by a distinct node label that does not occur anywhere in Σ. Clearly, $G \models \Sigma$, since there are no attributes in G that agree on their values. Thus, Σ does not prevent path Q/Q', as desired.

Now assume that $|[\![Q/Q']\!]| = 0$ in T_f. Then we show that $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$, which leads to a contradiction. Observe that the only possible children of $n \in [\![Q/Q']\!]$ are its required attributes. Since the removal of a required attribute always includes its parent, a single removal operation in the process is sufficient for removing n. Let us denote by T the tree obtained after executing a removal operation. We show that each possible removal corresponds to the application of certain rules of $\mathcal{I}_{att}$, and thus if $|[\![Q/Q']\!]| = 0$ then $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$.

We first consider the removal when Q'_ϕ is not of the form $P/@l$, as illustrate in Figure 2(b). Observe that since there exist two distinct nodes x, y in $[\![Q_\phi/Q'_\phi]\!]$ then in T there exist paths Q_c, Q_t, Q'_t, Q_r such that $Q/Q' = Q_c/Q_t/Q'_t/Q_r$, $Q_c \subseteq Q_\phi$, $Q_t \subseteq Q'_\phi$, and $Q_t/Q'_t \subseteq Q'_\phi$. Observe that since $x \neq y$, $Q'_t \neq \epsilon$. Thus, by *double-path* we obtain $(Q_c/Q_t/Q'_t//, S)$ from ϕ, and by *target-containment*, $(Q_c/Q_t/Q'_t/Q_r, S)$; that is, $(Q/Q', S)$. By *target-to-context*, $\Sigma \models (Q, (Q', S))$.

Now let us consider the case when $\phi = (Q_\phi, (Q''_\phi/@l, \emptyset))$. Let $Q/Q' = Q_c/Q_t/Q'_t/Q_r$, where $Q_c \subseteq Q_\phi$, $Q_t/@l \subseteq Q''_\phi/@l$, and $Q_t/Q'_t/@l \subseteq Q''_\phi/@l$. Observe that by construction, every attribute in T are those required to exist. That is, for every attribute node v, if $v \in [\![P/@a]\!]$ then $\Sigma \models (P, (\epsilon, \{@a\}))$ by *epsilon, superkey*, and the two *containment* rules. Thus, $\Sigma \models (Q_c/Q_t, (\epsilon, \{@l\}))$. The only exception is the existence of a possibly not required attribute $@a$ when φ is of the form $(Q, (Q'/@a, S))$. We distinguish between the following two cases:

(1) $@l$ is required to exist for nodes in $[\![Q_c/Q_t/Q'_t]\!]$, as illustrated in Figure 2(c): in this case, $\Sigma \models (Q_c/Q_t/Q'_t, (\epsilon, \{@l\}))$ and, by *double-attribute*, we can derive $(Q_c/Q_t/Q'_t//, S)$. By *target-containment*, we obtain $(Q_c/Q_t/Q'_t/Q_r, S)$; that is, $(Q/Q', S)$, and by *target-to-context*, $\Sigma \models (Q, (Q', S))$.

(2) $@l$ is not required to exist for nodes in $[\![Q_c/Q_t/Q'_t]\!]$, as illustrated in Figure 2(d): in this case, since $Q_t/Q'_t/@l \subseteq Q''_\phi/@l$, by *attribute-on-a-path* we obtain $(Q/Q_c/Q_t/Q'_t/@l, \emptyset)$; that is, $(Q/Q', \emptyset)$. Thus, by *target-to-context*, $\Sigma \models (Q, (Q', \emptyset))$.

This shows that, given key $\varphi = (Q, (Q', S))$, if Σ prevents path Q/Q' then $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$.

Part 2. The proof of the second part has mostly already been given in [2]. We include it to make the proof self-contained. Suppose that Σ *does not prevent* Q/Q' and $\Sigma \not\vdash_{\mathcal{I}_{att}} \varphi$. We show that $\Sigma \not\models \varphi$ by constructing an XML tree G such that $G \models \Sigma$ but $G \not\models \varphi$. In other words, if $\Sigma \models \varphi$ then $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$. The construction of G involves the following steps: First, we define an abstract tree T such that $T \not\models \varphi$. Then, T is modified in a way that the resulting tree T_f satisfies Σ. That is, for each $\phi \in \Sigma$, we check whether T satisfies ϕ. If not, certain nodes in T are merged such that the modified tree satisfies ϕ. At the end of the process, $T_f \models \Sigma$ and either: (1) $T_f \not\models \varphi$, and by Lemma 2.2, we can construct an XML tree G from T_f that satisfies Σ but not φ; or (2) $T_f \models \varphi$. In this case, we show that our assumption that $\Sigma \not\vdash_{\mathcal{I}_{att}} \varphi$ does not hold. That is, we show that each step of the merging operations corresponds to the application of certain rules in $\mathcal{I}_{att}$. Therefore, if $T_f \models \varphi$ then $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$, which contradicts the assumption.

We start by giving the construction of an abstract tree T that does not satisfy φ. As shown in Figure 3(a), T consists of a single path Q from the root leading to a node n, which has two distinct Q' paths leading to nodes n_1 and n_2. Next, for each element v in T we add all the attributes required by Σ. That is, v has attributes $\{@a \mid v \in [\![Q]\!],\ (Q_1, (Q_1', S)) \in \Sigma,\ Q \subseteq Q_1/Q_1',\ @a \in S\}$. A distinct value is assigned to each attribute created if $v \notin n[\![Q'/@A_i]\!]$, $i \in [1, k]$. In contrast, for each $i \in [1, k]$, let $val(n_1.@A_i) = val(n_2.@A_i)$. Let x_i denote the $@A_i$ attribute of n_1, and y_i the $@A_i$ attribute of n_2. Then $val(x_i) = val(y_i)$.

Note that if $Q' = \epsilon$, then $n = n_1 = n_2$, and therefore $T \models \varphi$ if and only if for all $@A_i$, $i \in [1, k]$, $@A_i$ exists for n. Then, we would have had $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$ by the *epsilon* and *superkey* rules in $\mathcal{I}_{att}$, a contradiction to our initial assumption that $\Sigma \not\vdash_{\mathcal{I}_{att}} \varphi$. Thus in the rest of the proof we assume that $Q' \neq \epsilon$. It is easy to see that in this case, $T \models \neg\varphi$.

We next modify T such that $T \models \Sigma$. We examine each ϕ in Σ, and if the abstract tree does not satisfy ϕ, then we merge certain nodes in the tree such that the modified tree satisfies ϕ. More specifically, let $\phi = (Q_\phi, (Q'_\phi, \{@A'_1, \ldots, @A'_m\}))$. We modify T as follows.

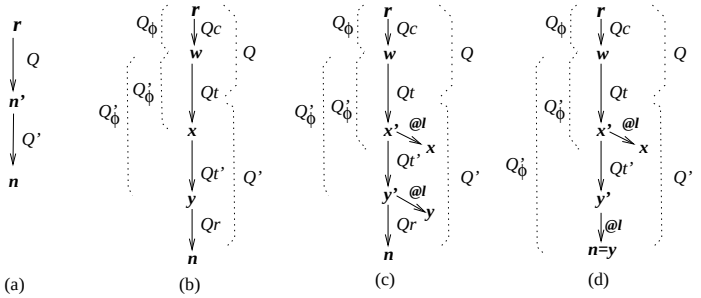

Figure 2: Abstract trees constructed in the proof of Theorem 2.1 - Part 1

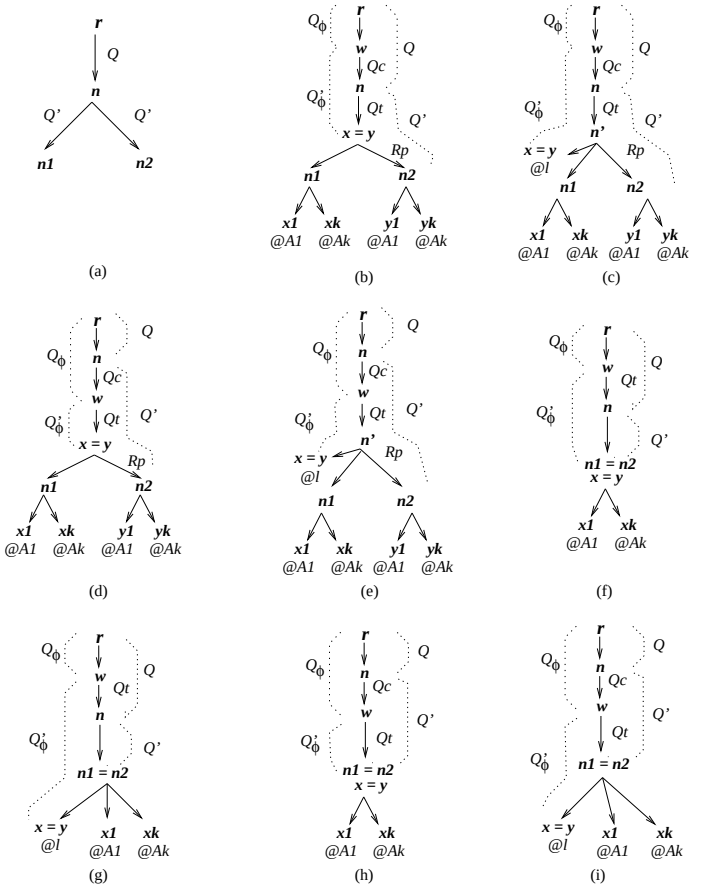

Figure 3: Abstract trees constructed in the proof of Theorem 2.1 - Part 2

repeat until no further change in T

if there exist key $\phi \in \Sigma$, node x with children $x'_1, \ldots, x'_m$, node y with children $y'_1, \ldots, y'_m$, and node w in T such that
$T \models Q_\phi(r, w) \wedge Q'_\phi(w, x) \wedge Q'_\phi(w, y) \wedge$
$@A'_1(x, x'_1) \wedge \ldots \wedge @A'_m(x, x'_m) \wedge$
$@A'_1(y, y'_1) \wedge \ldots \wedge @A'_m(y, y'_m) \wedge$
$val(x'_1) = val(y'_1) \wedge \ldots \wedge val(x'_m) = val(y'_m) \wedge x \neq y$
then merge x, y and their ancestors in T as follows:

Case 1: if x, y are above n_1, n_2 in T, that is, if Q_x is the sequence of labels from the root to x and $Q_x = P$ or $Q_x = P/@l$ for some proper prefix P of Q/Q' then merge nodes based on the position of w as shown in Figure 3 (b), (c), (d) and (e)

Case 2: otherwise, merge nodes as shown in Figure 3 (f), (g), (h) and (i);

By the construction of T, $val(x'_i) = val(y'_i)$ if and only if they correspond to attributes of n_1 and n_2, respectively. The process terminates since T is finite and thus merging can be performed only finitely many times. Note that T_f, the tree obtained upon termination of the merging process, satisfies Σ. We now examine whether or not $T_f \models \varphi$.

If $T_f \not\models \varphi$ then by Lemma 2.2, there is an XML tree G such that $G \models \Sigma$ and $G \not\models \varphi$, as desired. Note that $T_f \not\models \varphi$ if either $n_1 \neq n_2$ or there exists an attribute $@A_i$ ($i \in [1, k]$) missing for n_1 (and n_2).

Now assume that $T_f \models \varphi$. We show that $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$, which leads to a contradiction. Denote by T the tree obtained after executing z merging operations. We show by induction on z that each merging step corresponds to the application of

rules of $\mathcal{I}_{att}$, and thus if $T \models \varphi$ then $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$.

For the base case, $z = 0$, the statement holds since the initial tree does not satisfy φ. Assume the statement for z. We show that it also holds for $z+1$.

First, consider the merging in Case 1 as shown in Figure 3 (b), (c), (d) and (e). Since distinct values are assigned to attributes $@a$ not in φ, Case 1 can only happen if $m = 0$, that is, the set of key attributes in the key ϕ considered is empty. We consider the following cases.

(1) Node w is on the path Q, i.e., it is above n in T. Then there exist path expressions Q_t, R_p such that $Q' = Q_t/R_p$, and either (i) $x, y \in n[\![Q_t]\!]$ as illustrated in Figure 3(b), or (ii) $x, y \in n[\![Q_t/@l]\!]$ as shown in Figure 3(c). In this case, however, from ϕ, the key $(Q,\ (Q_t,\ \emptyset))$ can be proved, by using *target-to-context*, the two *containment* rules (i.e., *context-path-containment*, and *target-path-containment*), *superkey*, and *uniqueness*.

(2) Node w is on the path Q', i.e., it is below n but above n_1, n_2 in T. Then there exist path expressions Q_c, Q_t, R_p such that $Q/Q_c \subseteq Q_\phi$, $Q' = Q_c/Q_t/R_p$, $w \in n[\![Q_c]\!]$ and either (i) $x, y \in n[\![Q_c/Q_t]\!]$ as illustrated in Figure 3(d), or (ii) $x, y \in n[\![Q_c/Q_t/@l]\!]$ as illustrated in Figure 3 (e). This can only happen when some descendants x', y' of n on path Q' and above x, y were merged in a previous step by the process. More precisely, there are path expressions Q_{t1}, Q_{t2} such that $Q_t = Q_{t1}/Q_{t2}$, $x', y' \in n[\![Q_c/Q_{t1}]\!]$ and x', y' were previously merged. Thus by the induction hypothesis, we have that $(Q,\ (Q_c/Q_{t1},\ \emptyset))$ is provable from Σ by using $\mathcal{I}_{att}$. Furthermore, from ϕ one can prove $(Q/Q_c,\ (Q_{t1}/Q_{t2},\ \emptyset))$, by using the two *containment* rules, and *uniqueness*. Thus by *target-to-context* and *context-to-target* we have $(Q,\ (Q_c/Q_{t1}/Q_{t2},\ \emptyset))$, i.e., $(Q,\ (Q_c/Q_t,\ \emptyset))$.

We next consider the merging in Case 2 as shown in Figure 3 (f), (g), (h) and (i). By the definition of abstract trees, Case 2 can only happen if either $Q/Q' \subseteq Q_\phi/Q'_\phi$ or $Q/Q'/@l \subseteq Q_\phi/Q'_\phi$ and moreover, for all $j \in [1, m]$, there is $s \in [1, k]$ such that $@A_s = @A'_j$. There are two cases.

(1) Node w is on the path Q, i.e., it is above n in T. Then there exists path expression Q_t such that either (i) $Q_t/Q' \subseteq Q'_\phi$, and $x = y = n_1 = n_2$ as depicted in Figure 3(f), or (ii) $Q_t/Q'/@l \subseteq Q'_\phi$ as shown in Figure 3(g). In both cases φ can be proved from ϕ by using *target-to-context*, the two *containment* rules, *superkey* and *uniqueness*. Thus $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$.

(2) Node w is on the path Q', i.e., it is below n but above n_1, n_2 in T. Then there exist path expressions Q_c, Q_t such that $Q/Q_c \subseteq Q_\phi$, $Q' = Q_c/Q_t$, $w \in n[\![Q_c]\!]$, and either (i) $x = y = n_1 = n_2$ as illustrated in Figure 3(h), or (ii) $n_1 = n_2$ and x, y in $n_1[\![@l]\!]$ as illustrated in Figure 3(i). This can only happen when some descendants x', y' of n on path Q' above n_1, n_2 were merged in a previous step by the process. More precisely, there are path expressions Q_{t1}, Q_{t2} such that $Q_t = Q_{t1}/Q_{t2}$, $x', y' \in n[\![Q_c/Q_{t1}]\!]$ and x', y' were previously merged. Thus, by the induction hypothesis, $(Q,\ (Q_c/Q_{t1},\ \emptyset))$ is provable from Σ by using $\mathcal{I}_{att}$. From ϕ one can verify $(Q/Q_c, (Q_{t1}/Q_{t2}, \{@A_1,\ \ldots,\ @A_k\}))$ by using the two *containment* rules, *superkey*, and *uniqueness*. Thus by *context-to-target* and *target-to-context* we have $(Q,\ (Q_c/Q_{t1}/Q_{t2},\ \{@A_1,\ \ldots,\ @A_k\}))$, i.e., $(Q,\ (Q',\ \{@A_1,\ \ldots,\ @A_k\})) = \varphi$. Thus again $\Sigma \vdash_{\mathcal{I}_{att}} \varphi$, contradicting our assumption.

This shows that $\mathcal{I}_{att}$ is complete for implication analysis of XML keys in $\mathcal{K}_{att}$, the class of keys studied in [2]. □

2.2 Algorithm

Theorem 2.3: *Given a set $\Sigma \cup \{\varphi\}$ of XML keys of $\mathcal{K}_{att}$, whether $\Sigma \models \varphi$ can be decided in $O(|\Sigma|^2\ |\varphi|^2)$ time, where $|\Sigma|$, $|\varphi|$ are the sizes of Σ and φ, respectively.* □

We prove the theorem by providing an algorithm, referred to as `implication`, that decides whether $\Sigma \models \varphi$ in $O(|\Sigma|^2\ |\varphi|^2)$ time. The algorithm is an extension of the implication-checking algorithm given in [2], by including derivations of nonexistence keys. Indeed, except the part that involves function `prevent`, the algorithm is almost identical to its counterpart given in [2].

Algorithm `implication`, shown in Figure 4, is based on the inference rules of $\mathcal{I}_{att}$. In a nutshell, given a set Σ of $\mathcal{K}_{att}$ keys and a single $\mathcal{K}_{att}$ key $\varphi = (Q, (Q', S))$, the algorithm first extends Σ by including keys of the form $(Q_1, (Q'_1, \emptyset))$ derivable from Σ based on the *uniqueness* rule (Lines 1 to 2). Then it checks whether φ is derivable from Σ by a nonexistence key imposed by Σ, by invoking function `prevent` (Line 3). If this is not the case, that is, Σ does not prevent path Q/Q', then the algorithm checks whether or not φ can be obtained from Σ by "value-based" inference rules. It rewrites φ to a "normal form" (Lines 5 to 6), and it then checks whether or not $\Sigma \vdash_{\mathcal{I}_{att}} \varphi'$ (Lines 7 to 13).

The "normalization" step is to compute another key $\varphi' = (Q/P_1, (P'_1, S))$ from φ by pulling up a sub-path P_1 of the target path Q' of φ to its context, such that (1) $Q' = P_1/P'_1$, (2) if $\Sigma \models \varphi'$ then $\Sigma \models \varphi$, and (3) P_1 is the "longest" sub-path of Q' satisfying (1) and (2). Intuitively, we can always derive keys that are local to subtrees given that they hold on larger trees, but not the other way around. Thus the normalization is to rewrite φ to be "as local as possible". More specifically, the normalization is conducted by checking each key of the form $(Q_2, (Q'_2, \emptyset))$ in Σ, finding out whether $Q' = P_1/P'_1$, $P_1 \subseteq Q'_2$ and $Q \subseteq Q_2$, and if so, then rewriting φ to $\varphi' = (Q/P_1, (P'_1, S))$ by pulling the longest such P_1 of Q' to the context (Lines 5 to 6). Note that if $\Sigma \models \varphi'$ then $\Sigma \models \varphi$ by the *context-to-target* rule. Observe that this rule is applicable only if $\Sigma \models (Q, (P_1, \emptyset))$. Thus, before the normalization is conducted the algorithm first extends Σ by including keys of the form $(Q, (Q', \emptyset))$ derivable from Σ based on the *uniqueness* rule (Lines 1 to 2). This is done by invoking a function `exist`.

After the normalization, the algorithm checks whether $\varphi' = (Q_1, (Q'_1, S))$ is provable from Σ. Since the *context-to-target* rule has already been applied, and all keys that can be derived by the *uniqueness* rule are already in Σ, $\Sigma \models \varphi'$ if and only if one of the following holds:

(1) the target path Q'_1 is a single attribute label (Line 7) and $\Sigma \models \varphi'$ by the *attribute* rule;

(2) the target path $Q'_1 = \epsilon$ and all attributes in S are required to exist (Line 8); in this case, $\Sigma \models \varphi'$ by the *epsilon* rule. Observe that Function `exist` is invoked for checking the existence of attributes in S;

(3) there exists a key $\phi = (Q_2, (Q'_2, S_2))$ in Σ such that

(a) $S_2 \subseteq S_1$, and attributes in $(S_1 \setminus S_2)$ are required to exist (Line 10), and moreover,

(b) Q_1 of φ' can be partitioned into P_1/P'_1 such that $P_1 \subseteq Q_2$ and $P'_1/Q'_1 \subseteq Q'_2$ (Line 11).

Here condition (a) checks whether φ' can be derived from ϕ by the *superkey* rule, and (b) checks for the applica-

bility of *target-to-context*, *context-containment* and *target-containment*.

Function `prevent` checks whether φ can be derived from Σ by applying the new inference rules for nonexistence keys. Given $\varphi = (Q, (Q', S))$, it determines whether Q/Q' is prevented by Σ as follows. First, an XML tree T is constructed such that T contains a single path Q/Q', and every occurrence of "//" is replaced by a distinct node label that does not occur anywhere in Σ. Let r be the root, n be the leaf node, and $P(n_1, n_2)$ denote the path from node n_1 to n_2 in T. Based on T, function `prevent` returns `true` if there exists $\phi = (Q_1, (Q_2, \emptyset))$ in Σ such that one of the following holds:

(a) Q_2 is not of the form $Q_2'/@a$, and there exist nodes $w \in [\![Q_1]\!]$, $x \in w[\![Q_2]\!]$, $y \in w[\![Q_2]\!]$, such that $x \neq y$; in this case $\phi \vdash_{\mathcal{I}_{att}} \varphi$ by *double-path*, *target-to-context*, and the two containment rules;

(b) Q_2 is of the form $Q_2'/@a$, and there exist nodes $w \in [\![Q_1]\!]$, $x \in w[\![Q_2']\!]$, $y \in w[\![Q_2']\!]$, such that x is an ancestor of y, and attribute $@a$ is required to exist for x; in this case $\phi \vdash_{\mathcal{I}_{att}} \varphi$ by *attribute-on-a-path*, *target-to-context*, and the two *containment* rules if $P(y, n) = @a$; otherwise, if attribute $@a$ is required for y then $\phi \vdash_{\mathcal{I}_{att}} \varphi$ by *double-attribute*, *target-to-context*, and the *containment* rules.

Thus the algorithm checks if each of the rules in $\mathcal{I}_{att}$ can be applied in proving that $\Sigma \models \varphi$.

Complexity. Algorithm `implication` takes $O(|\Sigma|^2\ |\varphi|^2)$ time. Indeed, there is an $O(|P_1||P_2|)$-time algorithm [1] for checking if a path expression P_1 is contained in P_2. Thus Function `exist` is in $O(|\Sigma|(|Q| + |S|))$. Given this, Lines 1 to 2 of the algorithm take at most $O(|\Sigma|^2)$ time.

Function `prevent` is in $O(|\Sigma|^2|Q|^2)$. The construction of the XML tree takes $O(|Q|)$ time. Given that Q_1, Q_2' are core XPath queries and can be evaluated in $O(|T||Q|)$ time [3], Line 4 takes at most $O(|\phi||Q|^2)$ time. To see this, observe that there exist at most $|Q|$ nodes in $[\![Q_1]\!]$, and for each of these nodes, $w[\![Q_2']\!]$ is computed. Thus, the computation of w, x, y takes at most $O(|\phi||Q|^2)$. Since function `exist` takes $O(|\Sigma||Q|)$ time, the cost of Lines 3 to 6 is $O(|\phi||Q|^2 + |\Sigma||Q|)$. Given that there exists at most $|\Sigma|$ keys, the overall cost of function `prevent` is $O(|\Sigma|^2|Q|^2)$. Given this, Line 3 of the `implication` algorithm takes $O(|\Sigma|^2|\varphi|^2)$.

Line 5 takes at most $O(|Q_1'|\ (|Q_1||Q_2| + |P_1||Q_2'|))$ time since there exists $|Q_1'|$ ways of partitioning Q_1' in P_1 and P_1'. Therefore, for all keys in Σ, Lines 4 to 6 take at most $O(|\varphi|\ (|\varphi||\Sigma| + |\varphi||\Sigma|))$ time, that is $O(|\varphi|^2|\Sigma|)$. For Line 10, since S_2 and S_1 consist of simple attributes, the containment of S_2 in S_1 can be checked in $O(|S_2||S_1|)$ time, and the computation of $S_1 \setminus S_2$ takes at most $O(|S_1||S_2|)$ time with a result of size at most $O(|S_1|)$. Therefore, the invocation of Function `exist` takes $O(|\Sigma|(|Q_1|+(|Q_1'|+|S_1|)))$ time. Thus, Lines 10 to 11 take at most $O(|S_2||S_1| + |\Sigma|(|Q_1| + |Q_1'| + |S_1|) + |Q_1|\ (|P_1||Q_2| + (|P_1'| + |Q_1'|)|Q_2'|))$ time. Hence for all keys in Σ, Lines 9 to 12 cost at most $O((|\Sigma||\varphi|)+|\Sigma||\varphi|+|\varphi|\ (|\varphi||\Sigma|+|\varphi||\Sigma|))$ time, which again is in $O(|\Sigma|\ |\varphi|^2)$. Taken together, the algorithm takes at most $O(|\Sigma|^2 + O(|\Sigma|^2|\varphi|^2) + |\Sigma|\ |\varphi|^2)$ time, or simply $O(|\Sigma|^2\ |\varphi|^2)$.

This tells us that the implication analysis of XML keys of [2] remains in PTIME even if nonexistence keys are taken into account. In fact, the bound is precisely the same as the one given in [2]. It is possible to lower the $O(|\Sigma|^2\ |\varphi|^2)$ bound, by extending the analysis of "witness graph" of [4], which in turn is an extension of the abstract-tree idea given above. We omit this possible improvement to keep the algorithm as close to its counterpart given in [2] as possible.

Algorithm `implication`

Input: a set of XML keys Σ, and an XML key φ.
Output: `true` iff $\Sigma \models \varphi$.

1. for each key $\phi = (Q_2, (Q_2'/@l, \emptyset))$ in Σ do
2. if `exist`$(Q_2/Q_2', \{@l\})$ then $\Sigma := \Sigma \cup \{(Q_2, (Q_2', \emptyset))\}$;
3. if $\varphi = (Q, (Q', S))$ and `prevent`(Q/Q') then return `true`;
4. for each key $\phi = (Q_2, (Q_2', \emptyset))$ in Σ do
5. if $\varphi = (Q_1, (Q_1', S_1))$ and $Q_1 \subseteq Q_2$, and there is (longest) P_1 such that $Q_1' = P_1/P_1'$ and $P_1 \subseteq Q_2'$
6. then $\varphi := (Q_1/P_1, (P_1', S_1))$
7. if $\varphi = (Q, (@l, \emptyset))$ then return `true`
8. else if $\varphi = (Q, (\epsilon, S))$ then return `exist`(Q, S);
9. for each key $\phi = (Q_2, (Q_2', S_2))$ in Σ do
10. if $\varphi = (Q_1, (Q_1', S_1))$ and $S_2 \subseteq S_1$ and `exist`$(Q_1/Q_1', S_1 \setminus S_2)$ then
11. if there are P_1, P_1' such that $Q_1 = P_1/P_1'$, and $P_1 \subseteq Q_2$ and $P_1'/Q_1' \subseteq Q_2'$
12. then return `true`;
13. return `false`;

function `exist` (Q, S)

Input: Q: path expression; S: a set of attributes.
Output: `true` iff for each $l \in S$ and each $n \in [\![Q]\!]$, $n.@l$ exists.

1. $X := S$;
2. for each key $\phi = (Q_1, (Q_1', S_1))$ in Σ do
3. if $Q \subseteq Q_1/Q_1'$
4. then $X := X \setminus S_1$;
5. return $(X = \emptyset)$;

function `prevent` (Q)

Input: Q: path expression;
Output: `true` iff Σ prevents path Q.

1. T:= XML tree for Q/Q' with root r and leaf n;
2. for each key $(Q_1, (Q_2, \emptyset))$ in Σ do
3. if Q_2 is of the form $Q_2'/@a$ then
4. for each w, x, y in T such that $w \in [\![Q_1]\!]$, $x \in w[\![Q_2']\!]$, $y \in w[\![Q_2']\!]$, x is an ancestor of y, and `exist`$(P(r, x), @a)$ do
5. if $P(y, n) = @a$ then return `true`
6. else if `exist`$(P(r, y), \{@a\})$ then return `true`
7. else if there exist nodes w, x, y in T such that $w \in [\![Q_1]\!]$, $x \in w[\![Q_2]\!]$, $y \in w[\![Q_2]\!]$, and $x \neq y$ then
8. return `true`;
9. return `false`;

Figure 4: Algorithm for checking XML key implication

3. References

[1] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Reasoning about keys for XML. *Information Systems*, 28(8):1037–1063, 2003.

[2] S. Davidson, W. Fan, and C. Hara. Propagating XML constraints to relations. *Journal of Computer and System Sciences (JCSS)*, 73(3):316–361, May 2007.

[3] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing xpath queries. *ACM Transactions on Database Systems*, 30(2):444–491, June 2005.

[4] S. Hartmann and S. Link. Unlocking keys for XML trees. In *Proc. of Int'l Conf. on Database Theory (ICDT)*, pages 104–118, 2007.