Repairing Inconsistent Merged XML Data

Wilfred Ng

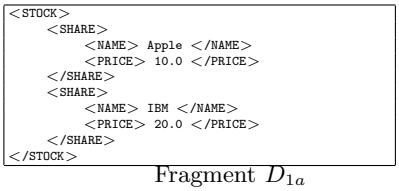
Department of Computer Science
The Hong Kong University of Science and Technology
Hong Kong, China
wilfred@cs.ust.hk

Abstract. XML is rapidly becoming one of the most adopted standard for information representation and interchange over the Internet. With the proliferation of mobile devices of communication such as palmtop computers in recent years, there has been growing numbers of web applications that generate tremendous amount of XML data transmitted via the Internet. We therefore need to investigate an effective means to handle such ever-growing XML data in various merging activities such as aggregation, accumulation or updating, in addition to storing and querying XML data. Previously, we recognized that FDs are an important and effective means to achieve consistent XML data merging, which we restricted data consistency for leaf nodes in an XML data tree. In this paper we further extend FDs to be satisfied in an XML document by comparing subtrees in a specified context of an XML tree. Given an XML tree T and a set of FDs F defined over a set of given path expressions, called targeted functional path expressions, we tackle the problem of repairing the inconsistency with respect to F in the most concise merged format of T.

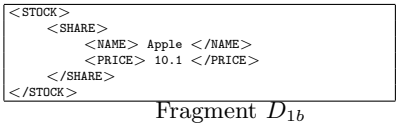
1 Introduction

There is now little debate that XML (eXtensible Markup Language) [1] will play an ever increasing role in web data specification and exchange. This increasing use of XML in web data specification and interchange increases the need for better tools and techniques to maintain the ever-growing XML data. Merging can be applied to XML documents in order to aid appending XML fragments/documents, accumulating XML data streams and to combining partial results for incremental querying. In this paper we develop the notion of merged XML trees and extend conventional FDs [7] being satisfied in XML trees in order to generate the most concise merged XML trees.

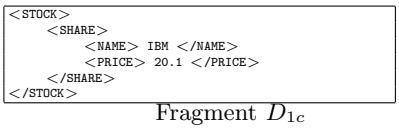
We follow the assumptions in our previous work (c.f. for details see [8]) when discussing the application of FDs to maintain data consistency in an XML setting and assume that DTD is absent in XML documents and that a pre-defined set of element labels (or tag names in common terms) is available to "spell" the path languages. The contribution of this work is that we generalise the notions of merged XML trees in [8] and extend the semantics of an FD being satisfied in an XML tree in order to cater for the situation when information are obtained from different XML data sources and therefore merging is necessary.



Fragment D_{1a}

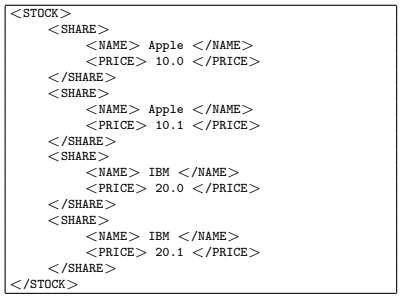


Fragment D_{1b}

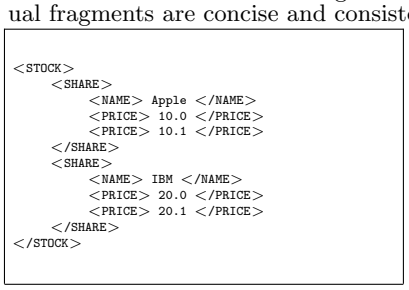


Fragment D_{1c}

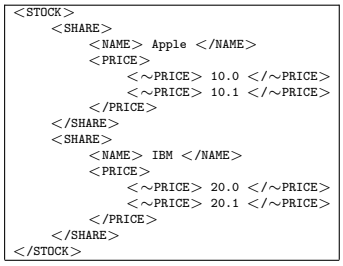
(a) A document is formed by direct combining the document fragments D_{1a}, D_{1b}, D_{1c} − such naive merging is not concise and inconsistent although individual fragments are concise and consistent.



(b) Document D_2 − a more concise merging at the STOCK level but the result is still inconsistent.



(c) Document D_3 − an even more concise merging at the SHARE level than D_2 but still inconsistent with respect to g.



(d) Document D_4 − an even more concise merging at the PRICE level and is consistent with respect to both f and g.

Fig. 1. Satisfaction of FDs in XML documents.

As a motivating example consider an XML document formed by a naive merging of three XML fragments D_{1a}, D_{1b} and D_{1c} given in Figure 1(a), which has $STOCK$ as a *context* node, and (share) name and (share) price as data nodes, all of these nodes are represented as their corresponding tags in D_1. In addition, we suppose that the FDs $f = SHARE.NAME \rightarrow SHARE$ and $g = SHARE.NAME \rightarrow SHARE.PRICE$ are specified as constraints, implying that *each share name has a unique piece of share information* and *each share has a unique price in the context of stock*. We assume that the current price information may be obtained from the three different XML data fragments D_{1a}, D_{1b} and D_{1c} as shown in Figure 1(a). Thus at any given time the naive *merging* of such information may be inconsistent. We now suppose that at some later stage the three fragments are merged at the STOCK level to be the document D_2 as shown in 1(b). It can be easily verified that D_2 satisfy neither f nor g, and is therefore inconsistent. However, it is *less verbose* and thus it is a *more concise* representation than a direct merging of D_{1a}, D_{1b} and D_{1c}. These fragments can

be merged at the SHARE level to generate an even more concise representation D_3, which is consistent with respect to f but is still inconsistent with respect to g, since we have more than one price element for each share. We propose to further fix this problem by "degenerating" the price node as shown in Figure 1. It can be easily verified that D_4 satisfies both f and g, and therefore the document is thus consistent. We will justify that the document D_3 is in fact the most concise representation of merged XML documents D_{1a}, D_{1b} and D_{1c} with respect to f, and D_4 is in fact the most concise representation of merged XML documents D_{1a}, D_{1b} and D_{1c} with respect to *both* f *and* g.

We adopt the usual notation in set theory and organise the rest of the paper as follows. In Section 2 we formalise the notion of merging in XML trees. In Section 3 we define the semantics of a functional dependency being satisfied in a merged XML tree and adapt the classical chase procedure in the context of XML, which is called XChase, for maintaining the consistency of an XML tree. In Section 4 we establish the notion of repairing and show that the output of XChase is always the consistent and most concise for merging a set of XML trees. Finally, in Section 5 we give our concluding remarks.

2 Merged XML Documents

In this section we formalise the notion of a merged XML document. We adopt the usual view that a document D is modeled as a node-labeled data tree T_D (or simply T whenever D is understood). In the sequel, we assume that (1) only two disjoint finite sets of labels exist in T as follows: $\mathbf{E}$ being the set of labels for element nodes (i.e. tag names) and S being a label denoting text nodes (i.e. PCDATA in XML documents), (2) an element node is either followed by a sequence of element nodes or is terminated with a text node, and (3) a countably infinite set of nodes $\mathbf{V}$ and a countably infinite domain of strings $\mathbf{S}$ exist.

Definition 1. (Merged XML Tree) A *merged XML node-labeled data tree* T_D corresponding to an XML document D (or simply an XML tree T) is defined by $T = (V, \lambda, \eta, \delta, r)$, where (1) $V \subseteq \mathbf{V}$ is a finite set of nodes; (2) λ is a mapping $V \rightarrow \mathbf{E} \cup \{S\}$ which assigns a label l to each node n in V; a node n in V is called an *element node* if $\lambda(n) \in \mathbf{E}$, and a *text node* if $\lambda(n) = S$; (3) η is a partial mapping that defines the edge relation of T as follows: if n is an element node in V then $\eta(n)$ is either a set of element nodes or a set of text nodes in V, and if n is a text node then $\eta(n)$ is undefined; and for each $n' \in \eta(n)$, n' is called a child of n, and we say that there is an edge from n to n', and n in V is called a *leaf node* if n' is a text node; (4) δ is a partial mapping that assigns a string to each text node: for any node n in V, if n is a text node then $\delta(n) \in \mathbf{S}$, and $\delta(n)$ is undefined otherwise; (5) $r \in V$ is a unique and distinguished *root* node.

It follows from Definition 1 that when given an XML tree for each $n \in V$, there is a *unique set of edges* (or *paths*) from root r to n. In addition, XML trees are finite due to the fact that V contains a finite number of nodes. In subsequent discussion, the terms XML documents and XML trees will be used

interchangeably, though we remark that the order of tags in an XML document is ignored in a tree representation according to Definition 1. Given T and $n \in V$ we denote by $\gamma(n)$ the subtree of T, which takes n as the root. Clearly, we have the special case $\gamma(r) = T$. We now introduce the useful notions related to comparing two subtrees in our context.

Definition 2. (Equality, Overlapping and Containment of XML Subtrees) Let $\gamma(r_1) = (V_1, \lambda_1, \eta_1, \delta_1, r_1)$ and $\gamma(r_2) = (V_2, \lambda_2, \eta_2, \delta_2, r_2)$ be two XML subtrees of a given tree T. Let $V_i \subseteq V$ be the set of nodes in $\gamma(n_i)$; λ_i, η_i and δ_i be the mappings *restricted* on the domain V_i for $i \in \{1, 2\}$. The XML trees $\gamma(r_1)$ and $\gamma(r_2)$ is said to be *equal*, denoted as $\gamma(r_1) \approx \gamma(r_2)$, if there exists a bijective mapping ρ from V_1 to V_2 satisfying that $r_2 = \rho(r_1)$, and for all $n'_1 \in V_1$, $\lambda_1(n'_1) = \lambda_2(\rho(n'_1))$, $\rho(\eta_1(n'_1)) = \eta_2(\rho(n'_1))$ and $(\delta_1(n'_1)) = \delta_2(\rho(n'_1))$. We say $\gamma(r_1)$ and $\gamma(r_2)$ are *overlapping* if there exists nodes $m_1 \in \eta_1(r_1)$, $m_2 \in \eta_2(r_2)$ such that $\gamma(m_1) \approx \gamma(m_2)$, or else we say $\gamma(r_1)$ and $\gamma(r_2)$ are *non-overlapping*. We say $\gamma(r_1)$ *contains* $\gamma(r_2)$ if for all nodes $m_2 \in \eta_2(r_2)$, there exists $m_1 \in \eta_1(r_1)$ such that $\gamma(m_1) \approx \gamma(m_2)$, or equivalently $\gamma(r_2)$ is contained in $\gamma(r_1)$.

Informally, two subtrees in T are equal if they are (1) isomorphic in structures and (2) identical in their corresponding element names and leaf values. Two subtrees are overlapping if they have equal children as their immediate subtrees. In our approach we consider only the immediate children for comparison, since we use the functional path as a means to specify the necessary depth involved, which will be explained in detail in Section 3. We also remark that using Definition 2 we can compare element nodes based on their subtree equality, overlapping and containment. In the special case of leaf nodes being the roots of the subtrees, the equality in our definition reduces to the equality of their respective sets of data values given by δ. Trivially, if two subtrees $\gamma(r_1)$ and $\gamma(r_2)$ are equal then they are overlapping. However, the converse may not be true.

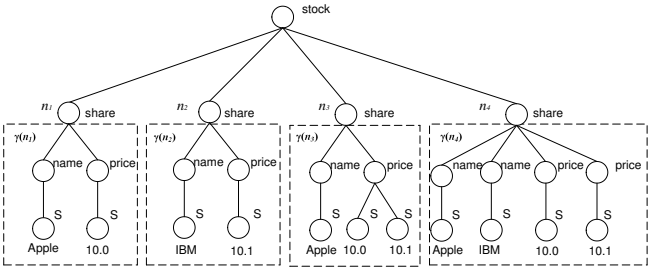


Fig. 2. Overlapping and containment of financial data

Example 1. In Figure 2, the subtrees $\gamma(n_1)$ and $\gamma(n_2)$ are non-overlapping, the subtrees $\gamma(n_1)$ and $\gamma(n_3)$ are overlapping since they have common children of name nodes, and the subtree $\gamma(n_4)$ contains both $\gamma(n_1)$ and $\gamma(n_2)$.

Definition 3. (Mutating a Subtree and Joining of Two Subtrees) Let $\gamma(r_1) = (V_1, \lambda_1, \eta_1, \delta_1, r_1)$ and $\gamma(r_2) = (V_2, \lambda_2, \eta_2, \delta_2, r_2)$ be two trees of T.

Given a subtree $\gamma(r_1)$, we define a *mutate operator*, denoted as μ, to generate a new subtree which is equal to $\gamma(r_1)$ but has a distinct set of nodes from those in $\gamma(r_1)$. Formally, $\mu(\gamma(r_1)) = (V_3, \lambda_3, \eta_3, \delta_3, r_3)$ such that $V_1 \cap V_3 = \emptyset$ and there exists a bijective mapping ρ from V_3 to V_1 satisfying that for all $n'_1 \in V_3$, $\lambda_3 = \lambda_1 \circ \rho$, $\eta_3 = \rho^{-1} \circ \eta_1 \circ \rho$ and $\delta_3 = \delta_1 \circ \rho$. We define the *join* of two distinct subtrees $\gamma(r_1)$ and $\gamma(r_2)$ (where $r_1 \neq r_2$) by $\gamma(r_1) \sqcup \gamma(r_2) = (V_1 \cup V'_2, \lambda_1 \cup \lambda'_2, \eta_1 \cup \eta'_2, \delta_1 \cup \delta'_2, r_1)$ where $\mu(\gamma(r_2)) = (V'_2, \lambda'_2, \eta'_2, \delta'_2, r_1)$ and $V_1 \cap V'_2 = r_1$.

The join operation provides us a basis for merging trees with respect to Ω which will be elaborated later on. Note that the mutation operation is necessary for technical reasons since the nodes used by $\gamma(r_2)$ cannot be used again in merging with $\gamma(r_1)$ according to Definition 3. The join operation essentially collects the children the under the two given subtrees under the root of the first subtree. Referring to Figure 2 it can be easily checked that $\gamma(n_4) = \gamma(n_1) \sqcup \gamma(n_2)$. The following propositions can be deduced from Definition

Proposition 1. The following statements are true.

1. $\mu(\gamma(r_1)) \approx \gamma(r_1)$.
2. $\gamma(x_1) \sqcup \gamma(x_2) \approx \gamma(x_2) \sqcup \gamma(x_1)$.
3. $\gamma(x_1) \sqcup (\gamma(x_2) \sqcup \gamma(x_3)) \approx (\gamma(x_1) \sqcup \gamma(x_2)) \sqcup \gamma(x_3)$. $\square$

We also need to use the concepts of a path expression, reachable nodes and the most distant node set (c.f. for details see [8], which are fundamental concepts prior to formalise the notion of XML document merging and the semantics of FDs in the context of XML trees. Essentially, a path expression in an XML tree T is essentially a special class of *regular expression*, which specifies a set of paths in T. For example, $node(FINANCE, STOCK.SHARE.NAME)$ means all the nodes of share names and $node(FINANCE, *.NAME)$ means all the nodes of (share, currency or brokers) names in the $FINANCE$ tree. In Figure 2, we can verify that all the $NAME$ nodes of shares are reachable from the root $FINANCE$ by following the path expression $STOCK.SHARE.NAME$ and all the $NAME$ nodes of currency are reachable from the root $FINANCE$ by following the path expression: $FOREIGN\_EXCHANGE.CURRENCY.NAME$.

3 Chasing XML Trees

We formalise the notion of an FD being satisfied in an XML tree, which is evolved from the one proposed in [8] as follows. We allow FDs to be defined for non-leaf nodes of an XML tree.

A *functional dependency* (FD) f over an XML tree T is a statement written in a triplet as follows, $(Q, Q', \{P_1, \ldots, P_m\} \rightarrow \{P_{m+1}, \ldots, P_n\})$, where Q, Q' and P_i are definite path expressions such that, for all $1 \leq i \leq n$, $Q.Q'.P_i$ is a valid path expression. The expression Q is called the *context path expression*, which specifies a set of paths starting from r; the expression $Q.Q'$ is called the *target path expression*, which specifies a set of paths within the context

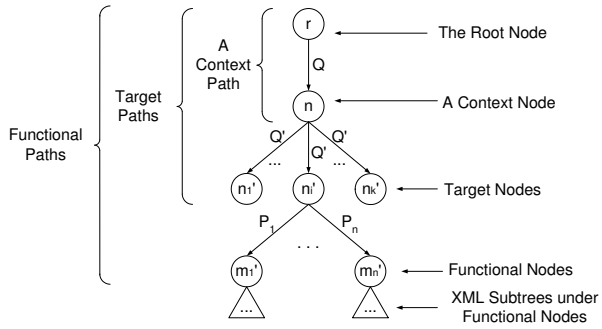


Fig. 3. Path expressions for reasoning FDs in an XML tree

following Q; and finally the expression $Q.Q'.P_i$ is called the *functional path expression*. We emphasize that a functional path expression specifies a set of paths which can reach functional nodes. We denote the *targeted functional path set* $\Omega(Q, Q')$ the collection of all possible P_i that follows the path specified the target path expressions $Q.Q'$ (i.e. $\Omega(Q, Q') = \{P_i \mid Q.Q'.P_i$ is a functional path expression $\}$). We assume throughout the paper Q and Q' denoting the context path expression and the target path expression and simplify the notations of targeted functional path set as Ω and the FD f as $P_1 \cdots P_m \rightarrow P_{m+1} \cdots P_n$. Using different path terminologies we have introduced for an XML tree and shown in Figure 3, we now define the semantics of FDs in an XML tree T.

Definition 4. (Functional Dependency Satisfication) Let T be an XML tree, $X = P_1 \cdots P_m$ and $Y = P_{m+1} \cdots P_n$. Then the FD $X \rightarrow Y$ is satisfied in T (or alternatively holds in T), denoted by $T \models X \rightarrow Y$, if, for any node $n \in node(r, Q)$, for any $n'_1, n'_2 \in node(n, Q')$ such that (1) $node(n'_1, P_i)$ and $node(n'_2, P_i)$ are non-empty for $i \in I_m$, and (2) for any two functional nodes $x_1 \in node(n'_1, P_i)$ and $x_2 \in node(n'_2, P_i)$, such that $\gamma(x_1)$ and $\gamma(x_2)$ are non-overlapping, it is also the case that, for $j \in \{m+1, \ldots, n\}$, if $node(n'_1, P_j)$ and $node(n'_1, P_j)$ are non-empty, then for any two functional nodes $y_1 \in node(n'_1, P_j)$ and $y_2 \in node(n'_2, P_j)$, $\gamma(y_1)$ and $\gamma(y_2)$ are non-overlapping.

Informally, along any two target paths (not necessarily to be distinct) specified by $Q.Q'$ in T which reach nodes n'_1 and n'_2 respectively, whenever *all* the children subtrees that follow the paths $P_1 \cdots P_m$ starting from n'_1 and n'_2 exist and overlap, then there exists corresponding subtrees specified by the path $P_{m+1} \cdots P_n$ also overlap if they exist. Note that there are three essential differences between the satisfaction of FDs in the context of an XML tree and a usual relation. First, the validity of the constraint holds only within the scope following the context path specified by Q and thus is only localized within the region of the subtree under a given node in $node(r, Q)$. Second, the semantics of FDs take into account of the fact that some path specified by the expressions in XY may have none or more than one occurrences. Third, the comparison

is based on overlapping of subtrees under the corresponding functional nodes rather than equality of attribute values.

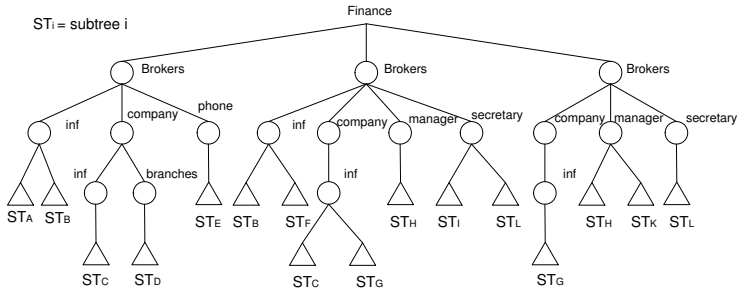


Fig. 4. An example of FD satisfaction in an XML tree

Example 2. In Figure 4, we show an XML tree (possibly) contains the information of a broker and the broker's company (labeled as *inf*), phone numbers, the corresponding details of the manager and the secretary. A company may have none or more than one secretary. It can be verified that the tree T with target node $BROKERS$ satisfies the following set of FDs: $\{NAME \rightarrow COMPANY.NAME, PHONE$; $COMPANY.NAME \rightarrow MANAGER$; $MANAGER \rightarrow SECERTARY\}$.

We now ready to formalise the concept of merging by making use of a targeted functional path set. Let $P \in \Omega$. For any $n \in node(r, Q)$ and any $n'_1, n'_2 \in node(n', Q')$ and for any two functional nodes $x_1 \in node(n'_1, P)$ and $x_2 \in node(n'_2, P)$, if there does not exist $r_1 \in \eta(x_1)$ and $r_2 \in \eta(x_2)$ such that $\gamma(r_1)$ and $\gamma(r_2)$ are overlapping, we define $\gamma(x_1)$ to be $\gamma(x_1) \sqcup \gamma(x_2)$ and $\gamma(x_2)$ to be $\gamma(x_2) \sqcup \gamma(x_1)$. We extend the joining of subtrees to be induced by the nodes in $node(n', P)$ for all $n' \in node(n, Q')$ and all $P \in \Omega$, and define a *merging* operation which replaces the children (i.e. the subtrees) following the functional paths specified by $Q.Q'.P$ by the joining of subtrees induced by $node(n', P)$.

Definition 5. (Merge Operation) The *merge* of an XML tree T with respect to a given Ω, denoted by $merge(T, \Omega)$, is the XML tree resulting from executing the following two steps.

1. **FOR** each $P \in \Omega$, any $n \in node(r, Q)$, and any pair of $n'_1, n'_2 \in node(n, Q')$, **WHILE** there are any two functional nodes $x_1 \in node(n'_1, P)$ and $x_2 \in node(n'_2, P)$ such that if $\gamma(x_1)$ and $\gamma(x_2)$ are overlapping, **DO** $\gamma(x_1) := \gamma(x_1) \sqcup \gamma(x_2)$ and $\gamma(x_2) := \gamma(x_2) \sqcup \gamma(x_1)$.
2. If there exists $n'_1, n'_2 \in node(n, Q')$ such that $\gamma(n'_1) \approx \gamma(n'_2)$ then remove $\gamma(n'_2)$.

Essentially, the first step in $merge(T, \Omega)$ replaces iteratively any pair of subtrees for functional nodes having the same expression by their merged set whenever an overlapping occurs. The second step remove some redundant subtrees

following $Q.Q'$. As an example, it can easily be verified that in Figure 5, the XML tree is $merge(T, \Omega)$ where $\Omega = \{CI$ (Company Information), PH (Price Histories)$\}$. The tree is obtained first by replacing the children of share names (CI and PH) by the join of their overlapping subtrees in the first step. Then, we remove the second occurrence of the share nodes in the second step, since they are duplicated under share nodes.

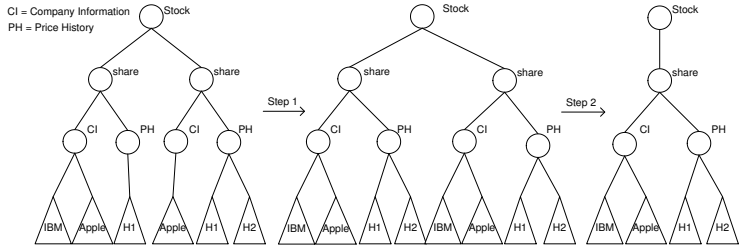


Fig. 5. An example of running the merge operation on an XML tree

Let $n \in node(r, Q)$. The XML tree resulting from $merge(T, \Omega)$ can be computed in polynomial time in the sizes of $node(n, Q')$, $node(r, Q)$ and Ω. We present in Algorithms 1 the pseudo-code for XChase(T, F), which is used for repairing inconsistency of an XML tree.

Algorithm 1 (XChase(T, F))

1. **begin**
2. Result := T ;
3. Tmp:= $\emptyset$;
4. **while** Tmp $\neq$ Result **do**
5. Tmp := Result;
6. **if** $\exists\ X \rightarrow Y \in F$, for any $n \in node(r, Q)$ where r is the root of Result, for any $n'_1, n'_2 \in node(n, Q')$ such that for all $i \in I_m$
   (1) $node(n'_1, P_i)$ and $node(n'_2, P_i)$ are non-empty, and
   (2) for $x_1 \in node(n'_1, P_i)$ and $x_2 \in node(n'_2, P_i)$,
   $\gamma(x_1)$ and $\gamma(x_2)$ being overlapping,
   but $\exists j \in \{m+1, \ldots, n\}$ such that $node(n'_1, P_j)$ and $node(n'_1, P_j)$ are non-empty, and for $y_1 \in node(n'_1, P_j)$ and $y_2 \in node(n'_2, P_j)$, $\gamma(y_1)$ and $\gamma(y_2)$ being non-overlapping
7. **then** $\gamma(y_1) := \gamma(y_1) \sqcup \gamma(y_2), \gamma(y_2) := \gamma(y_2) \sqcup \gamma(y_1)$;
8. **end while**
9. **return** $merge$(Result,Ω);
10. **end**.

Example 3. In Figure 6 we see that the XML tree T at (a) is inconsistent with respect to $F = \{P_1 \rightarrow P_2P_3, P_3 \rightarrow P_1\}$. We use the symbol $ST_{i_1,\ldots,i_n}$ to represent a set of the instances of n subtrees $ST_{i_1}, \ldots, ST_{i_n}$ under a functional node. The reader can also verify that the XML tree at (b) output from XChase(T, F) satisfies F, i.e. XChase(T, F) is consistent.

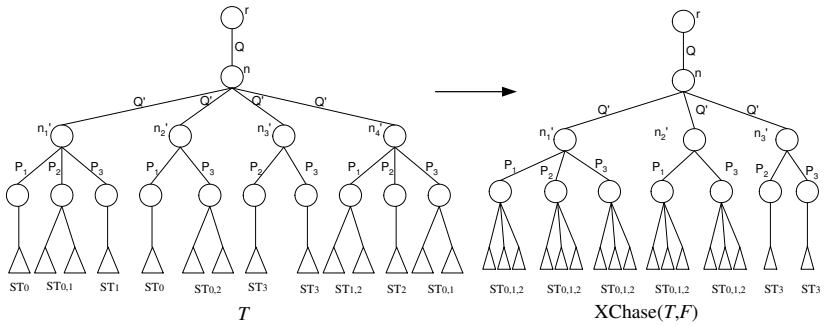


Fig. 6. An example of chasing an XML tree

XChase possesses the following desirable properties: (1) it outputs a consistent XML tree, (2) it is unique and can be computed in polynomial time in the sizes of T and F, and (3) it commutes with the merge operation. The next theorem shows that the chase procedure outputs a consistent XML tree and commutes with the merge operation.

Theorem 1. Let Ω be a targeted set of functional paths over T. Then the following statement is true. $\text{XChase}(T, F) = \text{XChase}(\mathit{merge}(T, \Omega), F)$. □

4 A Concise Semantic-Preserving XML Forests

In this section we assume throughout all XML trees are maximally merged, formally $T = \mathit{merge}(T, \Omega)$ and justify, by using a formal notion of repairing, that XChase generates the most concise merged XML tree that is semantic-preserving with respect to F. Using the notion of conciseness restricted to Ω we define an XML forest as a *conciseness-equivalent* class of XML trees and the *join* operation of two XML forests. Then we define the concepts of *semantic-preserving forest* for T with respect to a set of FDs F to be the *join* of all consistent XML forests which are less concise than the forest consisting T.

Definition 6. (Conciseness-Equivalent XML Trees) Let Ω be a targeted functional set. An XML tree T_1 is *less concise* than another XML tree T_2 with respect to Ω, written $T_1 \sqsubseteq_\Omega T_2$ (or simply $T_1 \sqsubseteq T_2$ whenever Ω are understood from the context), if, for any node $n_1' \in \mathit{node}(r_1, Q.Q')$ in T_1, there exists a node $n_2' \in \mathit{node}(r_2, Q.Q')$ in T_2 such that for all $i \in I_n$, for any two functional nodes $x_1 \in \mathit{node}(n_1', P_i)$ and $x_2 \in \mathit{node}(n_2', P_i)$, $\gamma(x_2)$ contains $\gamma(x_1)$. We say that T_1 and T_2 are *conciseness equivalent* with respect to Ω (or simply *Ω-equivalent*), written $T_1 \equiv_\Omega T_2$ (or simply $T_1 \equiv T_2$), if $T_1 \sqsubseteq_\Omega T_2$ and $T_1 \sqsubseteq_\Omega T_2$.

We call a collection of Ω-equivalent XML trees an *XML forest* with respect to Ω and denote it by $\Upsilon(\Omega)$. We say that T_1 *semantically preserves* T_2 if $T_2 \models f$

then $T_1 \models f$. The following result follows from Definition 6, which means our definition of conciseness preserve the semantics of XML data trees. The converse of the proposition is clearly not true, since two trees being mutually semantic-preserving may not be comparable with respect to $\sqsubseteq$.

Proposition 2. If $T_1 \equiv T_2$, then T_1 (or respectively T_2) semantically preserves T_2 (or respectively T_1). □

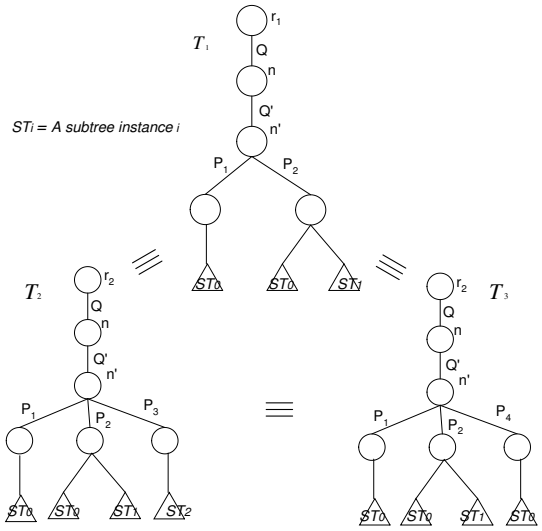


Fig. 7. A forest of Ω-equivalent trees

Figure 7 illustrates a simple forest having $\Omega = \{P_1, P_2\}$. Using the notion of conciseness, we are able to compare XML trees in some accurate sense and to define a semantic characterization of XChase later on. The following lemma shows that XChase result is less or equally concise as the original XML tree T.

Lemma 1. Let F be a set of FDs, Ω be a targeted functional set, and T be an XML tree. The statement $\text{XChase}(T, F) \sqsubseteq merge(T, \Omega)$ is true. □

Let $FOR(\Omega)$ be the collection of all XML forests w.r.t. Ω. The following proposition confirms that an XML forest is *uniformly concise and consistent*.

Proposition 3. Let $\Upsilon_1, \Upsilon_2 \in FOR(\Omega)$, $T_1, T_1' \in \Upsilon_1$, and $T_2, T_2' \in \Upsilon_2$. Then the following statements are true.

1. $T_1 \sqsubseteq T_2$ if and only if $T_1' \sqsubseteq T_2'$ for all $T_1' \in \Upsilon_1$ and $T_2' \in \Upsilon_2$.
2. $T_1 \models F$ if and only if $T_1' \models F$ for all $T_1' \in \Upsilon_1$. □

From Proposition 3 we are able to extend some concepts defined for an XML tree in Definition 6 to an XML forest. First, the partial order $\sqsubseteq_\Omega$ on $FOR(\Omega)$ is defined as follows: $\Upsilon_1 \sqsubseteq \Upsilon_2$ if $T_1 \sqsubseteq T_2$ where $T_1 \in \Upsilon_1$ and $T_2 \in \Upsilon_2$. Second, the satisfaction for an XML forest Υ with respect to F is defined by $\Upsilon \models F$ if, for all $T_1' \in \Upsilon_1$, $T_1' \models F$. We denote by $SAT(F)$ the set of all XML forests that satisfy

F. Finally, a merged XML forest is defined by $merge(\Upsilon, \Omega) = \{merge(T, \Omega) \mid T \in \Upsilon\}$. We now define the *join* operation on $FOR(\Omega)$ which returns a more concise forest than two given XML forests.

Definition 7. (Join of Two XML Forests) The *join* of two forests, $\Upsilon_1, \Upsilon_2 \in FOR(\Omega)$, denoted by $\Upsilon_1 \sqcup \Upsilon_2$, is defined by the set of XML trees that are Ω-equivalent to an XML tree T such that for any node $n'_1 \in node(r_1, Q.Q')$ in $T_1 \in merge(\Upsilon_1, \Omega)$, and $n'_2 \in node(r_2, Q.Q')$ in $T_2 \in merge(\Upsilon_2, \Omega)$, there exists a node $n' \in node(r, Q.Q')$ in T such that for all $P \in \Omega$, $\gamma(x) \equiv \gamma(x_1) \sqcup \gamma(x_2)$, where $x \in node(n, P)$, $x_1 \in node(n'_1, P)$ and $x_2 \in node(n'_2, P)$.

It is easy to see that the join of two merged XML forests is also a merged XML forest. The next theorem shows the fact that the join operation preserves the satisfaction of FDs.

Theorem 2. Let $\Upsilon_1, \Upsilon_2 \in SAT(F)$. Then $(\Upsilon_1 \sqcup \Upsilon_2) \in SAT(F)$. □

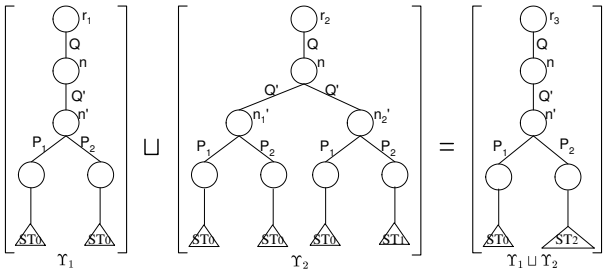


Fig. 8. A counter-example to the converse of Theorem 2

The converse of Theorem 2 is false as shown in Figure 8 where $ST_2 = ST_0 \sqcup ST_1$. Let $f = P_1 \rightarrow P_2$ be an FD. It can be verified that $\Upsilon_1 \sqcup \Upsilon_2 \models f$ but $\Upsilon_2 \not\models f$. We now formalise the concept of repairing an XML forest. A *repairing* for a given Υ is a less concise but still consistent forest. The *best possible repaired XML forest* for Υ with respect to a set of FDs F over Ω is the join of all consistent forests Υ' (i.e. $\Upsilon' \models F$) such that Υ' is less concise than Υ.

Definition 8. (Repaired XML Forest) Given Υ. A *repaired XML forest* for Υ with respect to F over Ω is an XML forest Υ' such that $\Upsilon' \sqsubseteq_\Omega \Upsilon$ and $\Upsilon' \in SAT(F)$. Let $REP(\Upsilon)$ be the set of all repaired XML forests for Υ. The *best possible repaired XML forest* for Υ with respect to F over Ω (or simply the repairing of Υ if F and Ω are understood from the context), denoted by $repair(\Upsilon, F)$, is given by $\bigsqcup_{\Upsilon' \in REP(F)} \Upsilon'$.

We note that the join of XML forests is well-defined, since the join operator is commutative and associative as shown in Proposition 1. The next theorem shows the main result in this section. It shows that the output of the XChase procedure is the best possible repairing, since the XML forest formed by the collection of all XML trees that are Ω-equivalent to XChase(T, F) is equal to $repair(\Upsilon, F)$, where $T \in \Upsilon$.

Theorem 3. Let $T \in \Upsilon$. Then XChase$(T, F) \in repair(\Upsilon, F)$. □

5 Concluding Remarks

We have defined the notion of a merged XML tree T with respect to a given set of FDs F over a targeted set of possible functional paths Ω specified by a contextual path Q and a target path Q' in T. The notion of an FD was extended to merged XML trees in Definition 4, whose satisfaction is defined by using overlapping immediate children subtrees of corresponding functional nodes over T, which is different from the usual way of defining FDs. We then defined XChase over XML trees in Algorithm 1 as a means of repairing inconsistency of an merged XML tree with respect to a set of FDs F. In addition, we established the desirable properties in Theorem 1 that XChase outputs a consistent XML tree and commutes with the merge operation. We further proposed to view the XML trees that are equivalent in conciseness over Ω as an XML forest in Definition 6 and defined the concept of the best possible repaired XML forest with respect to a set of FDs F in Definition 8. By using the join operation on merged XML forests we are able to show that if two XML forests are consistent then their join is also a consistent XML forest in Theorem 2. The best possible repaired XML forest for Υ with respect to F is the join of all consistent XML forests that are less concise than Υ, in this sense we presented our final result in Theorem 3, which shows that the chase procedure XChase(T, F) outputs the best possible repairing of T with respect to F.

Acknowledgements. This work is supported in part by grant HKUST 6185/02E from the Research Grant Council of Hong Kong.

References

1. S. Abiteboul, P. Buneman and D. Suciu. *Data on the Web.* Morgan Kaufmann Publishers, (2000).
2. P. Buneman, W. Fan, and S. Weinstein. *Interaction Between Path and Type Constraints.* Proc. of the 18th ACM Symposium on Principles of Database Systems (PODS'99), pp. 56-67, (1999).
3. Peter Buneman, Susan B. Davidson, Wenfei Fan, Carmem S. Hara, and Wang Chiew Tan. *Keys for XML.* Proc. of WWW10, pp. 201-210, (2001).
4. W. Fan, G.M. Kuper and J. Simon. *A Unified Constraint Model for XML.* Proc. of WWW10, pp. 179-190, (2001).
5. W. Fan and L. Libkin. *On XML Integrity Constraints in the Presence of DTDs.* Proc. of the 20th ACM Symposium on Principles of Database Systems, (PODS'01), (2001).
6. M. Levene and G. Loizou. Maintaining consistency of imprecise relations. *The Computer Journal* **39**, pp. 114-123, (1996).
7. H. Mannila and K-J Raiha. *The Design of Relational Databases.* Addison-Wesley, (1992).
8. W. Ng. *Maintaining Consistency of Integrated XML Trees.* LNCS Vol. 2419: Proc. of WAIM, Beijing, China, pp. 145 -157, (2002).
9. J. Wijsen. *Condensed Representation of Database Repairs for Consistent Query Answering* LNCS Vol. 2572: Proc. of 9th ICDT, Seina, Italy, pp. 378-393, (2002).