Updating in XML Using Semantic Constraints

Md. Sumon Shahriar ‡ and Jixue Liu†
‡ Tasmanian ICT Centre, CSIRO
Hobart 7001, Tasmania, Australia
† Data and Web Engineering Lab
School of Computer and Information Science
University of South Australia, Adelaide, SA-5095, Australia
E-mail: mdsumon.shahriar@csiro.au, jixue.liu@unisa.edu.au

Abstract

A novel technique for updating in XML using semantic constraints is proposed. In the proposed technique for XML updating, we consider close value pair semantics in ordered XML documents. Further, how semantic constraints can be incorporated with integrity constraints for XML updating and how to update XML views using semantic constraints are discussed.

Keywords: XML updating, semantic constraints, view updating, data transformation.

1. Introduction

Updating is an important operation in traditional database management systems [1]. In recent years, XML [2] is widely used in many database-centric applications such as data exchange [3] and data integration [4]. In database-centric approach of XML has necessitated the update operations in XML [5, 6] in recent past. Most update operations in XML are based on un-ordered semantics of XML data. However, updating XML using semantic constraints through ordered approach is little investigated to the best of our knowledge.

XML updating can be accomplished in two ways. First, updating can be done on the base XML document(s). Second, updating can be done on the materialized view derived from the base XML documents [7, 8, 9, 10, 11]. Without loss of generality, we say XML updating to mean the update operations on the base XML document and we say XML view updating to mean the update operations on XML materialized view. We present some motivating examples of XML updating.

Updating value of a node: Consider the DTD in Fig.1. The DTD gives the salary information of a person having first name (fname) and last name (lname).

```
<!ELEMENT db((fname, lname), salary)+>
<!ELEMENT fname(#PCDATA)>
<!ELEMENT lname(#PCDATA)>
<!ELEMENT salary(#PCDATA)>
```

Figure 1. An XML DTD
The XML document (in tree structure) in Fig. 2 conforms to the DTD in Fig. 1. We want to update the salary of the person having first name Sam and last name Andrew to 80K. First, we can update the node \( v_3 \) or the node \( v_6 \) labeled by salary considering the first name and the last name value pair for the nodes \( v_1 \) and \( v_5 \). Second, we can update the node \( v_9 \) labeled by salary considering the first name and the last name value pair for the nodes \( v_7 \) and \( v_8 \). We show the problem using question (?) marks in Fig. 3. Clearly, the first option is not semantically correct because the salary node \( v_3 \) is for the person having the first name Sam and the last name Kim and the salary node \( v_6 \) is for the person having the first name Kim and the last name Andrew.

**Adding a node:** Consider the DTD in Fig. 4. The DTD gives the information of name (first name \( \text{fname} \) and last name \( \text{lname} \)) and telephone numbers (\( \text{tell} \)).

```xml
<!ELEMENT info ((fname, lname); tell+)>
<!ELEMENT fname (#PCDATA)>
<!ELEMENT lname (#PCDATA)>
<!ELEMENT tell (#PCDATA)>
```

The XML document (in tree structure) in Fig. 5 conforms to the DTD in Fig. 4. We want to add one more telephone number 8334 to the person having first name John and last name Simpson. Now we have a problem in adding a node labeled \( \text{tell} \) with value 8334 in Fig. 6. We can add a node \( v_7 \) because the nodes \( v_4 \) and \( v_5 \) have the value pair (John, Simpson) or we can add a node \( v_{11} \) because the nodes \( v_8, v_9 \) have the same value pair. We see clearly that the
first value pair for nodes \( v_1 \) and \( v_5 \) is not semantically correct because the \( v_1 \) having value John is the first name of the person with the last name Smith of node \( v_2 \). The second value pair for the nodes \( v_8 \) and \( v_9 \) is semantically correct.

Deleting a node: Consider the DTD in Fig.7. It gives the enrollment information of students (with student id \( sid \)) in courses (with course id \( cid \)). The XML tree in Fig.8 conforms to the DTD in Fig.7.

\[
\begin{align*}
\langle &!ELEMENT enroll((sid; cid)+) > \\
\langle &!ELEMENT sid(#PCDATA) > \\
\langle &!ELEMENT cid(#PCDATA) > \\
\end{align*}
\]

Figure 7. An XML DTD

Now we want to delete a course having id CS04 for the student having id S001. Like previous examples, we have the problem of deleting either the node \( v_3 \) or the node \( v_5 \) (shown in Fig.9). Surely, the deletion of node \( v_3 \) is semantically correct.

We aim to contribute the followings.
(a) We propose semantic constraints for XML updating using close value pair concept for ordered XML documents.
(b) The close value pair concept can be incorporated with XML integrity constraints.
(c) We also study the XML view updating using semantics constraints.

This paper is organized as follows. In Section 2, we give basic definitions. We then study updating languages and give the algorithm for locating a position in the document to update in
2. Basic Definitions

In this section, we present basic definitions needed throughout the paper.

**Definition 2.1 (DTD)** An XML DTD is defined as $D = (EN, G, \beta, \rho)$ where

(a) $EN$ is a set of element names.

(b) $G$ is the set of type constructs and $g \in G$ is defined as one of the followings:
   (i) $g = Str$ where $Str$ means $\#PCDATA$;
   (ii) $g = e$ where $e \in EN$;
   (iii) $g = e$ means $EMPTY$ type;
   (iv) $g = g_1 \times g_1$ or $g_1 \mid g_1$ is called conjunctive or disjunctive sequence respectively where $g_1 = g$ is recursively defined, $g_1 \neq Str \land g_1 \neq e$; $g = g_2 \mid (g_2 = e \land e \in EN \lor g_2 = [g \cdots g] \lor g_2 = [g] \cdots [g])$, called a component where $c \in \{?, 1, +, *\}$ is the multiplicity of $g_2$, ‘[ ]’ is the component constructor;
   (c) $\beta(e) = [g]^c$ is the function defining the type of an element $e$ where $e \in EN$ and $g \in G$.
   We term $\beta(e)$ as element definition.
   (d) $\rho$ is the root of the DTD and that can only be used as $\beta(\rho)$. □

**Example 2.1** Consider the DTD in Fig.1. The DTD is defined as $D = \{EN, G, \beta, \rho\}$ where $EN = \{db, fname, lname, salary\}$, $G = \{[fname \times lname] \times salary]^+, \beta(db) = ([fname \times lname] \times salary]^+, \beta(fname) = Str, \beta(lname) = Str, \beta(salary) = Str$ and $\rho = db$.

We now define paths on the DTD.
Definition 2.2 (Path) Given a D = (EN, G, β, ρ), a simple path ϕ on D is a sequence e₁/⋯/eₘ, where Vi(eᵢ ∈ EN) and V eᵢ ∈ [ₑ₂, ⋯, eₘ] (eᵢ is an element in β(eᵢ₋₁)). A simple path ϕ is a complete path if e₁ = ρ. A path ϕ is empty if m = 0, denoted by ϕ = ε. When we say path, we mean simple path. We use function last(ϕ) to return eₘ, beg(ϕ) = e₁ and par(eᵢ) = eᵢ₋₁, the parent of eᵢ. We use len(ϕ) to return m.

Paths satisfying this definition are said valid on D. □

Example 2.2 Consider the DTD in Fig.1. In Figure, db/fname is simple path. db/fname is also a complete path because db is the root of the DTD. last(db/fname) is fname and beg(db/fname) is db. len(db/fname) is 2.

Definition 2.3 (XML Tree) The XML tree T representing an XML document is defined to be one of the followings:

(i) T = ϕ(empty).
(ii) A labeled node T = (v : e : txt).
(iii) If T₁, ⋯, Tₖ are trees, then T = (v : e T₁T₂⋯Tₖ) is also a tree. T₁⋯Tₖ are called subtrees.

The symbol v is the node identifier which can be omitted when the context is clear.

Example 2.3 Consider the document in Fig.2. The document is represented as Tᵥ₁ = (v₁ : fname : Sam), Tᵥ₂ = (v₂ : lname : Kim), Tᵥ₃ = (v₃ : salary : 60K), Tᵥ₄ = (v₄ : lname : Kim), Tᵥ₅ = (v₅ : lname : Andrew), Tᵥ₆ = (v₆ : salary : 90K), Tᵥ₇ = (v₇ : fname : Sam), Tᵥ₈ = (v₈ : lname : Andrew), Tᵥ₉ = (v₉ : fname : 75K).

Definition 2.4 (Hedge and Conformation) Let (v : eT₁⋯Tₙ) be a tree where T₁⋯Tₙ are subtrees of a node v with label e. A hedge H is a maximal sequence of consecutive subtrees from T₁⋯Tₙ that conforms to a component g under the context node v, denoted by H ∈ g or Hᵦ. When there are multiple hedges for g, we use Hᵢ to denote one of them and Hᵦ* to denote all of them.

Example 2.4 Consider the DTD in Fig.1 and the tree in Fig.2. Let the component be g₁ = [[fname×lname]×salary] be a component. Then H₁ᵠ₁ = Tᵥ₁Tᵥ₂Tᵥ₃, H₂ᵠ₁ = Tᵥ₄Tᵥ₅Tᵥ₆, and H₃ᵠ₁ = Tᵥ₇Tᵥ₈Tᵥ₉ are the hedges conforming to g₁. Let g₂ = [[fname×lname]×salary]* be the component. Then the hedge H₁ᵦ² = Tᵥ₁Tᵥ₂Tᵥ₃Tᵥ₄Tᵥ₅Tᵥ₆Tᵥ₇Tᵥ₈Tᵥ₉ is the only hedge conforming to g₂.

We define the tree conformation using the hedge.

Definition 2.5 (Tree Conformation) Given a DTD D = (EN, G, β, ρ) and XML Tree T, T conforms to D denoted by T ∈ D if T = (ρ Hᵦ(ρ)). □

Example 2.5 Consider the DTD in Fig.1 and the tree representation of the XML document in Fig.2. The root ρ = db and the the element definition β(ρ) = [[fname×lname]×salary]*. The hedge H[[fname×lname]×salary]* = Tᵥ₁Tᵥ₂Tᵥ₃Tᵥ₄, Tᵥ₅Tᵥ₆Tᵥ₇Tᵥ₈Tᵥ₉ conforms to the component [[fname×lname]×salary]*. We then get Tᵦ = (dbH[[fname×lname]×salary]*). Thus the tree in Fig.2 conforms to the DTD in Fig.1.
Definition 2.6 (Tree Equivalence) Two trees $T_a$ and $T_b$ are value equivalent, denoted by $T_a =_v T_b$, if

1. $T_a = (v_1 : e : txt1)$ and $T_b = (v_2 : e : txt1)$, or
2. $T_a = (v_1 : e T_1 \cdots T_m)$ and $T_b = (v_2 : e T'_1 \cdots T'_n)$ and $m = n$ and for $i = 1, \cdots, m(T_i =_v T'_i)$. \hfill \Box

$T_x = T_y$ if $T_x$ and $T_y$ refer to the same tree. We note that, if $T_x = T_y$, then $T_x =_v T_y$.

Example 2.6 Consider two trees $T_a = (v_1 : f \text{name} : \text{Sam})$ and $T_b = (v_7 : f \text{name} : \text{Sam})$. Then $T_a =_v T_b$.

Definition 2.7 (Hedge Equivalence) Two hedges $H_x$ and $H_y$ are value equivalent, denoted by $H_x =_v H_y$, if

1. Both $H_x$ and $H_y$ are empty, or
2. $H_x = T_1 \cdots T_m$ and $H_y = T'_1 \cdots T'_n$ and $m = n$ and for $i = 1, \cdots, m(T_i =_v T'_i)$. \hfill \Box

We define minimal structure of a DTD.

Definition 2.8 (Minimal Structure) Given an element definition $\beta(e)$ and two elements $e_1$ and $e_2$ in $\beta(e)$ of a DTD, the minimal structure $g$ of $e_1$ and $e_2$ in $\beta(e)$ is the innermost component belonging to the component $\beta(e)$ that contains both $e_1$ and $e_2$. \hfill \Box

Example 2.7 Let $db$ be an element and $[[f \text{name} \times l \text{name}] \times \text{salary}]^+$ be the element definition of $db$ in the DTD in Fig.1. Now consider two elements f name and salary. Then the minimal structure for the elements $f$ name and salary is $[[f \text{name} \times l \text{name}] \times \text{salary}]$. But the minimal structure of $f$ name and $l$ name is $[f \text{name} \times l \text{name}]$.

Definition 2.9 (Minimal Hedge) A hedge that conforms to a minimal structure $g$ is a minimal hedge. \hfill \Box

We now define prefixed trees. The reason for defining prefixed trees is to get the trees for a path.

Definition 2.10 (Prefixed Trees) We use $T^e$ to denote a tree rooted at a node labeled by the element name $e$. Given path $e_1/\cdots/e_m$, we use $(v_1 : e_1)\cdots(v_{m-1} : e_{m-1}).T^e_m$ to mean the tree $T^e_m$ with its ancestor nodes in sequence, called the prefixed tree or the prefixed format of $T^e_m$.

Given path $\varphi = e_1/\cdots/e_m$, $T^\varphi$ denotes $(v_1 : e_1)\cdots(v_{m-1} : e_{m-1}).T^e_m$ and is called a tree of the path $\varphi$. When the context is clear, we use $T^\varphi$ to mean $T^\varphi$.

Given path $\varphi = e_1/\cdots/e_k/\cdots/e_m$, $	ext{prec}(T^\varphi) = (v_1 : e_1)\cdots(v_k : e_k)\cdots(v_m : e_m)$ means the precision of the tree $T^\varphi$ for path $\varphi$.

$(T^\varphi)$ is the set of all $T^\varphi$. $(T^\varphi) = \{T^\varphi_1, \cdots, T^\varphi_f\}$. $|\langle T^\varphi \rangle|$ returns the number of $T^\varphi$ in $(T^\varphi)$.

Example 2.8 Consider an element salary. Then $T^\text{salary}$ means one of the trees $T_{v_3}^\text{salary}$ and $T_{v_9}^\text{salary}$ in Fig.2. Let $\varphi = db/salary$. The prefixed trees of $T^\text{salary}$ are $(v_r : db).T_{v_3}^\text{emp}$, $(v_r : db).T_{v_9}^\text{emp}$. The precisions of the tree $T^\varphi$ are $	ext{prec}(T^\varphi) = (v_r : db)$. Then $\langle T^\varphi \rangle = \left\langle T_{v_3}^\text{emp} \right\rangle = \{T_{v_3}, T_{v_9}, T_{v_9}\}$ and $|\langle T_{v_3}^\text{emp} \rangle| = 3$.

3. XML Updating

In the section, we study updating on XML. First, we define updating language.
3.1. Updating language

We now define the language for updating XML.

Update a node: We define the language of updating value of a node as: update $Q$ set $\text{val}(Q)$ where $P_1 = \text{val}(P_1) \cdots$ and/or $\cdots P_n = \text{val}(P_n)$.

We now give an example.

Example 3.1 Consider the DTD in Fig.1 and the conforming document in Fig.2. We want to update salary 75K of Sam Andrew to 80K. The language for this updating is: "update $db/salary$ set $\text{val}(db/fname) = \text{Sam and } db/lname = \text{Andrew}".

Add a node: We define the language of adding a node as: add $Q$ where $P_1 = \text{val}(P_1) \cdots$ and/or $\cdots P_n = \text{val}(P_n)$.

We now give an example.

Example 3.2 Consider the DTD in Fig.4 and the conforming document in Fig.5. We want to add a telephone number 8334 for John and Simpson. The language for this updating is: "add $\text{info/fname} = \text{John and } info/lname = \text{Simpson}".

Delete a node: We define the language of deleting a node as: delete $Q$ where $P_1 = \text{val}(P_1) \cdots$ and/or $\cdots P_n = \text{val}(P_n)$.

We now give an example.

Example 3.3 Consider the DTD in Fig.7 and the conforming document in Fig.8. We want to delete the course CS004 for the student id S001. The language for this updating is: "delete enroll/cid where enroll/cid = CS004 and enroll/sid = S001".

We first define the following definition.

Definition 3.1 (Close Value Pair) Given a set of complete paths $\{P_1, \ldots, P_l\}$. A close value pair is a sequence of pair-wise-close subtrees, denoted by $F[P] = (T^{P_1} \cdots T^{P_l})$, where 'pair-wise-close' is defined next.

Let $\varphi_i = e_1 / \cdots / e_k / e_{k+1} / \cdots / e_m \in P_i$ and $\varphi_j = e'_1 / \cdots / e'_k / e'_{k+1} / \cdots / e'_n \in P_j$ be two paths for any $P_i$ and $P_j$. Let $\text{prec}(T^{P_i}) = (v_1 : e_1), \ldots, (v_k : e_k), (v_{k+1} : e_{k+1}), \ldots, (v_m : e_m)$ and $\text{prec}(T^{P_j}) = (v'_1 : e'_1), \ldots, (v'_k : e'_k), (v'_{k+1} : e'_{k+1}), \ldots, (v'_n : e'_n)$. Then $T^{P_i}$ and $T^{P_j}$ are pair-wise-close if, $e_k = e'_k, \ldots, e_k = e'_k, e_{k+1} \neq e'_{k+1}$, then $v_k = v'_k$, $(v_{k+1} : e_{k+1})$ and $(v'_{k+1} : e'_{k+1})$ are two nodes in the same minimal hedge of $e_{k+1}$ and $e'_{k+1}$ in $\beta(e_k)$. □

Example 3.4 Consider the DTD in Fig.1 and the document in Fig.2. Consider the $db/fname$, $db/lname$ and $db/salary$ are the complete paths. We see that $db$ is common to all of these complete paths. Then minimal structure for the elements $fname$, $lname$ and $salary$ is $g_m = [\text{fname}, \text{lname}, \text{salary}]$. Then the minimal hedges are $H_{1m} = T_{v_1} T_{v_2} T_{v_3}$, $H_{2m} = T_{v_4} T_{v_5} T_{v_6}$ and $H_{3m} = T_{v_7} T_{v_8} T_{v_9}$. So (Sam, Kim, 60K) is the close pair value for the hedge $H_{1m}$. (Kim, Andrew, 90K) is the close pair value for the hedge $H_{2m}$ and (Sam, Andrew, 75K) is the close pair value for the hedge $H_{3m}$. However, the pair (Sam, Andrew, 90K) is not the close pair value. There are two possibilities for not being the close pair value. First, Sam is taken from hedge $H_1$ and Andrew, 90K are taken from $H_2$. Second, Sam, Andrew are taken from hedge $H_3$ and 90K is taken from $H_2$. 

International Journal of Database Theory and Application
Vol. 4, No. 2, June 2011
Example 3.5 Consider the example 3.3. In the example, \( Q = \text{enroll/cid} \), \( P_1 = \text{enroll/cid} \) and \( P_2 = \text{enroll/sid} \). So the minimal structure is \( g_{m1} = \{ \text{sid} \times \text{cid} \} \) in the DTD in Fig.7 and the minimal hedges are \( H_{g_{m1}} = T_{v_1}T_{v_2}T_{v_3} \) and \( H_{g_{m2}} = T_{v_4}T_{v_6}T_{v_5} \). The close value pairs are \( (T_{v_1}T_{v_2}) = (S001, CS02) \) and \( (T_{v_4}T_{v_5}) = (S007, CS04) \) for minimal hedge \( H_{g_{m1}} \) and \( (T_{v_4}T_{v_5}) = (S007, CS07) \) for minimal hedge \( H_{g_{m2}} \). The close value pair for \( Q, P \) is \( (S001, CS04) \). So we only delete the node \( T_{v_3} \) in Fig.8.

We present the algorithm 1 for updating XML.

**Input:** An XML document \( T \), DTD \( D \), update parameters \( Q, P \);

**Output:** Updated document \( T' \);

1. Calculate minimal structure \( g_{m} \) for \( Q, P \);
2. Calculate minimal hedges for \( g_{m} \);
3. Generate close value pairs for minimal hedges;
4. Search and locate the close value pair according to value pair for \( Q, P \);
5. Update the located close value pair;

**Algorithm 1:** UPDATE.XML.DOCUMENT(\( T, D, Q, P \))

4. XML Integrity Constraints with Semantics for Updating

We first define the close value pair equivalence.

**Definition 4.1 (Close Value Pair Equivalence)** Two close value pairs \( F_1[P] = (T_1^{p_1} \cdots T_l^{p_1}) \) and \( F_2[P] = (T_2^{p_1} \cdots T_k^{p_2}) \) are value equivalent, denoted by \( F_1[P] =_v F_2[P] \) if \( l = k \) and for each \( i = 1, \ldots, k \) \( T^{p_i}_1 =_v T^{p_i}_2 \). \( \square \)

![Figure 10. An XML tree](image)

**Example 4.1** Consider the XML document in Fig.10. The close value pairs \( (T_{v_1}T_{v_2}) \) and \( (T_{v_4}T_{v_5}) \) are value equivalent because \( T_{v_1} =_v T_{v_4} \) and \( T_{v_2} =_v T_{v_5} \).

In the XML document, there may be duplicate close value pairs meaning that any two close value pairs are equivalent. In that case, updating is done for duplicate close value pairs. If there is an XML key \( (Q, P) \) defined, meaning that the close value pairs are distinct, then we update only one close value pair.
5. Updating on XML Views

As XML has been widely accepted as a data model over the web and in many data-centric applications, the research on XML view updating [12] has also got significant attention to the database community. Most researches on XML view updating are on the XML views derived from data with schema based on relational data model. However, the researches on XML view updating where XML views are also derived from XML data (with or without schema) are little investigated to the best of our knowledge. The research in [10] considers view updating where both the base documents and the view document are in XML. The XML view updating in [10] considers the recursion in the base documents. Our research is different from the research in [10] in the sense that we do not consider recursion and we consider the ordered property in XML document which is not considered in [10].

![Figure 11. An XML DTD](image1.png)

![Figure 12. An XML tree](image2.png)

We already showed how to update an XML document that conforms to a DTD using semantic constraints by close value pair concept. We now study how to update an XML view. We first give a motivating example. Consider the DTD in Fig.11 and the conforming document in Fig.12. In the document, the first name and last name of persons and their home and office telephone numbers are stored. Now, we transform the DTD and the document (using either XQuery [13] or transformation operators [14]) to the DTD in Fig.13 and the document in Fig.14 as view.

Now we want to update the contact telephone number 6677 to 6999 of the person having first name Sam and last name Peter in the view document in Fig.14. The fname, lname and contact in the view document maps to the fname, lname and home in the base document in Fig.12.

However, we need to locate the correct person to update the contact at the view document as well as home telephone number in the base document. In this case, we need the close value pair mechanism to locate the correct place for updating.
6. Conclusions

We proposed a novel method of XML updating using close value pair in ordered XML documents. We addressed how semantic constraint expressed using close value pair can be incorporated with XML integrity constraints. Further, we discussed XML view updating problem using semantic constraints. We plan to study performance analysis of update operators for synthetic and real world data sets.

7. Acknowledgement

The Tasmanian ICT Centre is jointly funded by the Australian Government through the Intelligent Island Program and CSIRO. The Intelligent Island Program is administered by the Tasmanian Department of Economic Development and Tourism.

This research was conducted as part of the CSIRO Wealth from Oceans National Research Flagship and Data and Web Engineering Lab, Computer and Information Science School, University of South Australia.

This paper is the extended (invited) version of the conference paper published in the international conference on database theory and its applications (DTA 2010), South Korea [15].

References


Biography

![Biography Image]
Dr. Md. Sumon Shahriar: Sumon Shahriar is currently post doctoral fellow at Tasmanian ICT Centre, CSIRO Australia. He got his PhD in XML database from Data and Web Engineering Lab, School of Computer and Information Science, University of South Australia. He achieved his Bachelor of Science (Honours) and Master of Science (Research) degrees both with first class in Computer Science and Engineering from University of Dhaka, Bangladesh. His research interests include XML database, data integration, data quality and data mining.

![Photo of Dr. Md. Sumon Shahriar](image_url)

Dr. Jixue Liu: Jixue Liu got his bachelor’s degree in engineering from Xian University of Architecture and Technology in 1982, his Masters degree (by research) in engineering from Beijing University of Science and Technology in 1987, and his PhD in computer science from the University of South Australia in 2001. His research interests include view maintenance in data warehouses, XML integrity constraints and design, XML and relational data, constraints, and query translation, XML data integration and transformation, XML integrity constraints transformation and transition, and data privacy.