Update XML Views

Jixue Liu[1] Chengfei Liu[2] Theo Haerder[3] Jeffery Xu Yu[4]

[1] School of Computer and Information Science, University of South Australia
email: {jixue.liu}@unisa.edu.au

[2] Faculty of ICT, Swinburne University of Technology
email: cliu@swin.edu.au

[3] Dept of Computer Sciences, Technical University of Kaiserslautern
email: haerder@informatik.uni-kl.de

[4] Dept of Systems Eng. and Eng. Management, Chinese University of HK
email: yu@se.cuhk.edu.hk

Completed 2011 June

Abstract

View update is the problem of translating an update to a view to some updates to the source data of the view. In this paper, we show the factors determining XML view update translation, propose a translation procedure, and propose translated updates to the source document for different types of views. We further show that the translated updates are precise. The proposed solution makes it possible for users who do not have access privileges to the source data to update the source data via a view.

keywords: XML data, view update, update translation, virtual views

1 Introduction

A (virtual) view is defined with a query over some **s**ource data of a database. The query is called the **v**iew definition which determines what data appears in the view. The data of the view, called a **view instance**, is often not stored in the database but is derived from the source data on the fly using the view definition every time when the view is selected.

In database applications, many users do not have privileges to access all the data of a database. They are often given a view of the database so that they can retrieve only the data in the view. When these users need to update the data of the database, they put their updates against the view, not against the source data, and expect that the view instance is changed when it is accessed next time. This type of updates is called a **v**iew update. *Because of its important* *use*, view update has a long research history [1, 8, 10, 11, 5, 3, 12]. The work in [4] discusses detailed semantics of view updates in many scenarios.

Unfortunately, view updates cannot be directly applied to the view instance as it is not stored physically and is derived on the fly when required (virtual view). Even in the cases where the view instance is stored (materialized view), which is not the main focus of this paper, applying updates to the instance may cause inconsistencies between the source data and the instance. To apply a view update to a virtual view, a translation process is required to translate the view update to some *source updates*. When the source data is changed, the data in the view will be changed next time when the view is selected. To the user of the view, it seems that the view update has been successfully applied to the view instance.

Let V be a view definition, V^i the view instance, S^i the source data of the view, $V(S^i)$ the evaluation of V against S^i. Then $V^i = V(S^i)$. Assume that the user wants to apply a view update δV to V^i as $\delta V(V^i)$. View update translation is to find a process that takes V and δV as input and produces a source update δS to S^i such that next time when the user accesses the view, the view instance appears changed and is as expected by the user. That is, for any S^i and $V^i = V(S^i)$,

$$V(\delta S(S^i)) = \delta V(V^i) \tag{1}$$

Two typical anomalies, view side-effect and source document over-update, are easily introduced by the translation process although they are update policy dependent [8]. View side-effect [12] is the case where the translated source update causes more-than-necessary change to the source data which leads to more-than-expected change to the view instance. View side-effect makes Equation (1) violated.

Over-updates may also happen to a source document. An over-update to a source document causes the source data irrelevant to the view to be changed, but keeps the equation satisfied. A source document over-update is incorrect as it changes information that the user did not expect to change.

A **precise** translation of a view update should produce source updates that (1) result in necessary (as the user expects) change to the view instance, (2) do not cause view side-effect, and (3) do not cause over-updates to the source documents.

In relational databases, extensive work has been done on view update and the problem has been well understood [1, 8, 10]. In cases of updating XML views over relational databases, updates to XML views need to be translated to updates to the base relational tables. The works in [3, 12] propose two different approaches to the problem. The work in [3] translates an XML view to some relational views and an update to the XML view to updates to the relational views. It then uses the relational approach to derive updates to the base tables. The work in [12] derives a schema for the XML view and annotates the schema based on keys of relational tables and multiplicities. An algorithm is proposed to use the annotation to determine if a translation is possible and how the translation works. Both works assume keys, foreign keys and the join operator based on these two types of constraints. Another work, technical report [5], proposes brief work on updating hypertext views defined on relational databases. To the best of our knowledge, the only work relating to XML view update is [7] which proposes a middle language and a transformation system to derive view instance from source data, and to derive source data from a **materialized** view instance, and assumes XQuery as the view definition language. We argue that with the view update problem, only view updates are available but not the view instance (not materialized). Consequently view update techniques are still necessary.

In this paper, we look into the view update problem in pure XML context. This means that both source data and the view are in XML format. We assume that base XML documents have no schema and no constraints information available.

The view update problem in the relational database is already difficult as not all view updates are translatable. For example, if a view V is defined by a Cartesian product of two tables R and S, an update inserting a new tuple to the view instance is not translatable because there is no unique way to determine the change(s) to R and S. The view update problem in XML becomes much harder. The main reason is that the source data and view instances are modeled in trees and trees can nest in arbitrary levels. This fundamental difference makes the methods of translating view updates in the relational database not applicable to translating XML view updates. For example, the selection and the projection in the relational database do not have proper counterparts in XML. The view update problem in XML has many distinct cases that do not exist in the view update problem in the relational database (see Sections 3 and 5 for details). To the best of our knowledge, our work is the first proposing a solution to the view update problem in XML.

We notice that the view update problem is different from the view maintenance problem. The former aims to translate a view update to a virtual view to a source update while the latter aims to translate a source update to a view update to a materialized view. The methods for one do not work for the other.

We make the following contributions in this paper. Based on the view definition and the update language presented later, we identify the factors determining the view update problem. We propose a translation algorithm to translate view updates to source updates. Furthermore, we propose translated updates to the source for different types of view updates. The types of view updates range from the case where the update involves an individual tree selected the source, the case where the update involves multiple trees from the source, and the case where the update happens to the root of the view. For each proposed update to the source, we prove that it is precise.

The paper is organized as follows. Section 2 shows the view definition language, the update language, and the preciseness of view update translation. In Section 3, we propose an algorithm and show that the translation obtained by the algorithm is a precise translation. In Section 4, we identify a 'join' case where a translated update is precise. Section 5 shows a translation when a main subtree of the view is deleted. Section 6 concludes the paper.

2 Preliminaries

In this section, we define basic notation, introduce the languages for view definitions and updates, and define the XML view update problem.

Definition 1 (tree)**.** An XML document can be represented as an ordered tree. Each node of the tree has a unique identifier v_i, an element name ele also called a **label**, and either a text string txt or a sequence of child trees $T_{j_1}, \cdots, T_{j_n}$. That is, a node is either $(v_i : ele : txt)$ or $(v_i : ele : T_{j_1}, \cdots, T_{j_n})$. When the context is clear, some or all of the node identifiers of a tree may not present explicitly. A tree without all node identifiers is called a **value tree**. Two trees T_1 and T_2 are (value) **equal**, denoted by $T_1 = T_2$, if they have identical value trees. If a tree T_1 is a subtree in T_2, T_1 is said **in** T_2 and denoted by $T_1 \in T_2$. □

For example, the document `<root><A><B>1</B></A><A><B>2</B></A></root>` is represented by $T = (v_r\!:\!root\!:\ (v_0\!:\!A\!:\!(v_1\!:\!B\!:\!1)), (v_2\!:\!A\!:\!(v_3\!:\!B\!:\!2)))$. The value tree of T is $(root\!:\ (A\!:\!(B\!:\!1)), (A\!:\!(B\!:\!2)))$.

Definition 2. A **path** p is a sequence of element names $e_1/e_2/\cdots/e_n$ where all names are distinct. The function $L(p)$ returns the last element name e_n.

Given a path p and a sequence of nodes $v_1, \cdots, v_n$ in a tree, if for every node $v_i \in [v_2, \cdots, v_n]$, v_i is labeled by e_i and is a child of v_{i-1}, then $v_1/\cdots/v_n$ is a **doc path** conforming to p and the tree rooted at v_n is denoted by $T^p_{v_n}$. □

2.1 View definition language

We assume that a view is defined in a dialect of the *for-where-return* clauses of XQuery [2].

Definition 3 (V)**.** A view is defined by

$$
\begin{array}{l}
<v>\{\ \texttt{for}\ x_1\ \texttt{in}\ p_1,\quad \cdots,\quad x_n\ \texttt{in}\ p_n \\
\qquad \texttt{where}\ \ cdn(x_1, \cdots, x_n) \\
\qquad \texttt{return}\ rtn(x_1, \cdots, x_n)\quad \}</v>
\end{array}
$$

where $p_1, \cdots, p_n$ are paths (Definition 2) proceeded by $doc()$ or x_i;

$cdn(x_1, \cdots, x_n) ::= x_i/\mathcal{E}_i = x_j/\mathcal{E}_j$ and $\cdots$ and $x_k/\mathcal{E}_k = strVal$ and $\cdots$;

$rtn(x_1, \cdots, x_n) ::= <\mathfrak{e}>\ \{x_u/\gamma_u\} \cdots \{x_v/\gamma_v\}\ </\mathfrak{e}>$;

$\gamma, \mathcal{E}$ are paths, and the last elements of all $x_u/\gamma_u, \cdots, x_v/\gamma_v$ are distinct. □

We note that the paths in the *return* clause are denoted by x_i/γs because these expressions are specially important in view update translation. We purposely leave out the $ sign proceeding a variable in the XQuery language.

Definition 4 (context-based production)**.** By the formal semantics of XQuery [6], the semantics of the language is

```
for x_1 in p_1 return
   for x_2 in p_2 return
      ...
         for x_n in p_n return
            if cdn(x_1, ..., x_n)=true
               return rtn(x_1, ..., x_n)
```

The for-statement produces tuples $<x_1, ..., x_n>$, denoted by $fortup(V)$, where the variable x_i represents a binding out of the sub trees located by p_i within the context defined by $x_1, \cdots, x_{i-1}$. This process is called **context-based production**. $\Box$

For each tuple satisfying the condition $cdn(x_1, ..., x_n)$, the function $rtn(x_1, \cdots, x_n)$ produces a tree, called an $\mathfrak{e}$-tree, under the root node of the view. That is, V maps a tuple to an $\mathfrak{e}$-tree. The children of the $\mathfrak{e}$-tree are the γ**-trees** selected by all the expressions x_i/γ_is (for all i) from the tuple. A tuple is mapped to one and only one $\mathfrak{e}$-tree and an $\mathfrak{e}$-tree is for one and only one tuple. A γ-tree of a tuple is uniquely mapped to a child of the $\mathfrak{e}$-tree of the tuple and a child of an $\mathfrak{e}$-tree is for one and only one γ-tree of its tuple.

The path of a node s in the view has the following format:

$$v/\mathfrak{e}/\mathcal{L}_i/\theta_i \tag{2}$$

$$\mathcal{L}_i = L(x_i/\gamma_i) \tag{3}$$

where x_i/γ_i is an expression in $rtn(x_1, ..., x_n)$, $L(x_i/\gamma_i)$ returns the last element name $\mathcal{L}_i$ of the path x_i/γ_i, and θ_i is a path following $\mathcal{L}_i$ in the view. When $\mathcal{L}_i/\theta_i$ is not empty, the path in the source document corresponding to $v/\mathfrak{e}/\mathcal{L}_i/\theta_i$ is

$$x_i/\gamma_i/\theta_i$$

The view definition has some properties important to view update translation. Firstly because of context-based production, a binding of variable x_i may be copied into $x_i^{(1)}, \cdots, x_i^{(m)}$ to appear in multiple tuples:

$$\begin{array}{l} <\cdots, x_i^{(1)}, \cdots, x_{j[1]}, \cdots> \\ \cdots \\ <\cdots, x_i^{(m)}, \cdots, x_{j[m_j]}, \cdots> \end{array}$$

where $x_{j[1]}, \cdots, x_{j[m_j]}$ are different bindings of x_j. Each tuple satisfying the condition $cdn(x_1, \cdots, x_n)$ is used to build an $\mathfrak{e}$-tree. As a result of x_i being copied, the subtrees of x_i will be copied accordingly to appear in multiple $\mathfrak{e}$-trees in the view.

Secondly, a tree may have zero or many sub trees located by a given path p. That is, given a tree bound to x_i, the path expression x_i/p may locate zero or many sub trees $T_1^{x_i/p}, \cdots, T_{n_p}^{x_i/p}$ in x_i. This is true in the source documents and in the view.

Thirdly, two path expressions x_i/γ_i and x_j/γ_j generally may have the same last element name, i.e., $L(x_i/\gamma_i) = L(x_j/\gamma_j)$. For example, if x_i represents an employee while x_j represents a department, then $x_i/name$ and $x_j/name$ will present two types of names in the same $\mathfrak{e}$-tree. This make the semantics of the view data not clear. This is the reason that we assume that all $L(x_i/\gamma_i)$s are distinct.

Example 1. Consider the view definition below and the source document shown in Figure 1(a). The view instance is shown in Figure 1(b).

```
<v>{for x in doc("r")/r/A,  y in x/C,   z in x/H
     where y/D=z and z="1"
     return <𝔢>{x/B}{x/C}{y/F/G}{z}</𝔢>
 }</v>
```

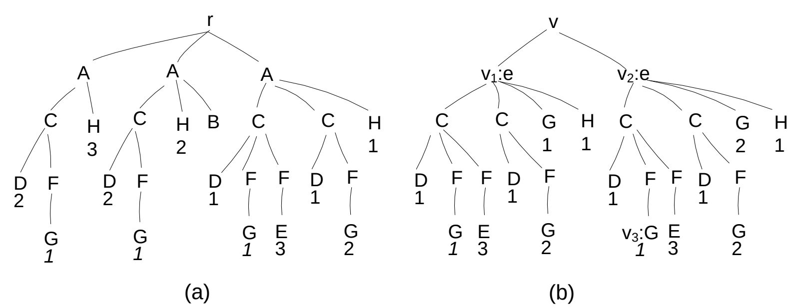

Figure 1: Source document r and view v

From the view definition, $\gamma_1 = B$, $\gamma_2 = C$, $\gamma_3 = F/G$, and $\gamma_4 = \phi$. $L(x/\gamma_1) = \mathcal{L}_1 = B$, $L(x/\gamma_2) = \mathcal{L}_2 = C$, $L(y/\gamma_3) = \mathcal{L}_3 = G$, and $L(z/\gamma_4) = \mathcal{L}_4 = H$.

Formula (2) is exemplified as the following. The node v_3 in the view has the path $v/e/C/F/G$ where C is $\mathcal{L}_2 = L(x/\gamma_2)$ and F/G is θ. The node v_1 is an $\mathfrak{e}$ node and its path is v/e and $\mathcal{L}_i/\theta_i$ is ϕ.

The example shows the following.

- The expression x/B $(=x/\gamma_1)$ of the *return* clause has no tree in the $\mathfrak{e}$-trees.
- The path expression x/C $(=x/\gamma_2)$ has multiple trees in an $\mathfrak{e}$-tree.
- The trees of x/C are duplicated in the view and so are their sub trees.
- Each of some x/C trees has more than one $x/C/F$ $(=x/\gamma_2/\theta)$ sub trees.

2.2 The update language

The update language we use follows the proposal [9] extended from XQuery.

Definition 5 (δV)**.** A view update statement has the format of

`for` $\bar{x}_1$ `in` $\bar{p}_1$`,` $\cdots$`,` $\bar{x}_u$ `in` $\bar{p}_u$
`where` $\bar{x}_c/\bar{p}_c = strValu$
`update` $\bar{x}_t/\bar{p}_t$ `(` `delete` T `|` `insert` T `)`

where $\bar{x}_c, \bar{x}_t \in [\bar{x}_1, \cdots, \bar{x}_u]$, $\bar{p}_1, \cdots, \bar{p}_u$ are paths (Definition 2) proceeded by v or $\bar{x}_i$; $\bar{p}_c, \bar{p}_t$ are paths; all element names in the paths are elements names in the view. $\bar{x}_c/\bar{p}_c$ and $\bar{x}_t/\bar{p}_t$ are called the **(update) condition path** and **(update) target path** respectively. □

The next process builds the mapping represented by Formula (3).

Procedure 1 (mapping)**.** When the variables in $\bar{x}_c/\bar{p}_c$ and $\bar{x}_t/\bar{p}_t$ are replaced by their paths in the for-clause until the first element name becomes v, the full paths of $\bar{x}_c/\bar{p}_c$ and $\bar{x}_t/\bar{p}_t$ will have the format of $v/\mathfrak{e}/\mathcal{L}_c/\theta_c$ and $v/\mathfrak{e}/\mathcal{L}_t/\theta_t$ as shown in Formula (2). The element names $\mathcal{L}_c$ and $\mathcal{L}_t$, if $\mathcal{L}_c/\theta_c$ and $\mathcal{A}_t/\theta_t$ are not empty, must be the last element names of two expressions x_c/γ_c and x_t/γ_t in the $return$ clause of the view definition V. A search using $\mathcal{L}_c$ and $\mathcal{L}_t$ in V will identify the expressions. Consequently $v/\mathfrak{e}/\mathcal{L}_c/\theta_c$ and $v/\mathfrak{e}/\mathcal{L}_t/\theta_t$ are mapped to $x_c/\gamma_c/\theta_c$ and $x_t/\gamma_t/\theta_t$ respectively. □

With this mapping, the update statement δV can be represented by the following abstract form:

$$(\bar{p}_s; \;\; v/\mathfrak{e}/\mathcal{L}_c/\theta_c = strValu; \;\; v/\mathfrak{e}/\mathcal{L}_t/\theta_t; \;\; del(T)|ins(T)) \tag{4}$$

where

- $v/\mathfrak{e}/\mathcal{L}_c/\theta_c$ is the **full** update condition path (int the view) for $\bar{x}_c/\bar{p}_c$, $v/\mathfrak{e}/\mathcal{L}_t/\theta_t$ the **full** target path for $\bar{x}_t/\bar{p}_t$;
- $\bar{p}_s$ is the maximal common front part of $v/\mathfrak{e}/\mathcal{L}_c/\theta_c$ and $v/\mathfrak{e}/\mathcal{L}_t/\theta_t$.

The semantics of an update statement is that under a context node identified by $\bar{p}_s$, if a sub tree identified by $v/\mathfrak{e}/\mathcal{L}_c/\theta_c$ satisfies the update condition, all the sub trees identified by $v/\mathfrak{e}/\mathcal{L}_t/\theta_t$ will be applied the update action (del(T) or ins(T)). The sub tree $T^{v/\mathfrak{e}/\mathcal{L}_c/\theta_c}$ is called the **condition tree** of $T^{v/\mathfrak{e}/\mathcal{L}_t/\theta_t}$. A sub tree is updated only if it has a condition tree and the condition tree satisfies the update condition. An update target and its condition trees are always within a tuple when the view definition is evaluated and are in an $\mathfrak{e}$-tree in the view after the evaluation.

We note that because of the context-based production in the update language, the same update action may be applied to a target node for multiple times. For example, if x is binding and the context-based production produces two tuple for it $< x^{(1)}, \cdots >$ and $< x^{(2)}, \cdots >$. If the update condition and target are all in x, x will be updated twice with the same action. We assume that only the effect of the first application is taken and the effect of all other applications are ignored.

Based on the structure of the target path $tp = v/\mathfrak{e}/\mathcal{L}_t/\theta_t$, updates may happen to different types of nodes in the view.

- When $\mathcal{L}_t/\theta_t \neq \phi$, the update happens to the nodes within a γ-tree.
- When $tp = v/\mathfrak{e}$, the update will add or delete a γ-tree.
- When $tp = v$ (in this case, $\bar{p}_s = v$), the update will add or delete an $\mathfrak{e}$-tree.

We will present the first case in Sections 3 and 4 and present the last two cases in Section 5.

2.3 The view update problem

Definition 6 (Precise Translation)**.** Let V be a view definition and S be the source of V. Let δV be an update statement to V. Let δS be the update statement to S translated from δV. δS is a precise translation of δV if, for any instance S^i of S and $V^i = V(S^i)$,

(1) δS is correct. That is, $V(\delta S(S^i)) == \delta V(V^i)$ is true; and

(2) δS is minimal. That is, there does not exist another translation $\delta S'$ such that ($\delta S'$ is correct, i.e., $V(\delta S'(S^i)) = V(\delta S(S^i)) = \delta V(V^i)$ and there exists a tree T in S^i and T is updated by δS but not $\delta S'$). □

We note that Condition (1) also means that the update δS will not cause view-side-effect. Otherwise, $V(\delta S(S^i))$ would contain more, less, or different updated trees than those in $\delta V(V^i)$.

Definition 7 (the view update problem)**.** Given a view V and a view update δV, the problem of view update is to (1) develop a translation process P, and show that the source update δS obtained from P is precise, or (2) prove that a precise translation of δV does not exist. □

3 Update Translation when $\mathcal{L}_t/\theta_t \neq \phi$ and $x_c = x_t$

In this section, we investigate update translation when the update is to change a γ-tree of the view and the mappings of the update condition path and the target path refer to the same variable. We present Algorithm 1 for view update translation in this case. The algorithm is self-explainable.

Algorithm 1: A translation algorithm

Input: view definition V, view update δV
Output: translated source update δS

begin
- make a copy of V and reference the copy by δS ;
- remove $rtn()$ from δS ;
- from the view update δV, following Procedure 1, find mappings $x_c/\gamma_c/\gamma_c$ and $x_t/\gamma_t/\gamma_t$ for the condition path $\bar{x}_c/\bar{p}_c$ and the target path $\bar{x}_t/\bar{p}_t$;
- make a copy of δV and reference the copy by δV_c ;
- in δV_c, replace $\bar{x}_c/\bar{p}_c$ and $\bar{x}_t/\bar{p}_t$ by $x_c/\gamma_c/\gamma_c$ and $x_t/\gamma_t/\gamma_t$ respectively ;
- append the condition in the *where* clause of δV_c to the end of the *where* clause in δS using logic *and* ;
- append the *update* clause of δV_c after the *where* clause of δS

By the algorithm, the following source update is derived.

$$
\begin{array}{ll}
\delta S: & \texttt{for } x_1 \texttt{ in } p_1, \quad \cdots, \quad x_n \texttt{ in } p_n \\
& \texttt{where } cdn(x_1, \cdots, x_n) \texttt{ and } x_c/\gamma_c/\theta_c = strValu \\
& \texttt{update } x_t/\gamma_t/\theta_t \quad \texttt{(insert } T \texttt{ | delete } T\texttt{)}
\end{array} \tag{5}
$$

We now develop the preciseness of the translation. We recall notation that $fortup(V)$ means the tuples of the context-based production (Definition 4) of V. $x_c^{(1)}$ and $x_c^{(2)}$ are two copies of a binding of x_c, and x_c, $x_{c[1]}$ and $x_{c[2]}$ are three separate bindings of x_c.

Lemma 1. *Given a tuple $t =< x_t, x_c, \cdots >\in fortup(V)$ and its $\mathfrak{e}$-tree e, (1) if T is a tree for the path $x_t/\gamma_t/\theta_t$ in t and T is updated by δS, then all the trees identified by $x_t/\gamma_t/\theta_t$ in t are updated by δS, and all the trees identified by $\mathcal{L}_t/\theta_t$ in e are updated by δV. (2) if T is a tree for the path $\mathcal{L}_t/\theta_t$ in e and T is updated by δV, then all the trees identified by $x_t/\gamma_t/\theta_t$ in t are updated by δS, and all the trees identified by $\mathcal{L}_t/\theta_t$ in e are updated by δV.*

The lemma is correct because of the one-to-one correspondences between a tuple and an $\mathfrak{e}$-tree and between t's γ-trees and e's children, and because all the trees identified by $x_t/\gamma_t/\theta_t$ in t share the same condition tree(ies) identified by $x_c/\gamma_c/\theta_c$ in x_c of t, and all the trees identified by $\mathcal{L}_t/\theta_t$ in e share the same condition tree(ies) identified by $\mathcal{L}_c/\theta_c$ in e.

Lemma 2. *Given a tuple $t =< x_t, x_c, \cdots >\in fortup(V)$, let a subtree $T^{x_t/\gamma_t/\theta_t}$ of x_t be updated by δS and become $t' =< x'_t, x_c, \cdots >$. If $x_t/\gamma_t/\theta_t$ is not a prefix of any of the path in the where clause of δS, if t satisfies $cdn()$ of V, t' also satisfies $cdn()$ of V.*

The lemma is correct because the subtrees in the tuple used to test $cdn()$ are not changed by δS when the condition of the lemma is met.

Lemma 3. *Given a tuple $t =< x_t, x_c, \cdots >\in fortup(V)$ and its $\mathfrak{e}$-tree e, if the $T^{x_c/\gamma_c/\theta_c}$ in t satisfies $x_c/\gamma_c/\theta_c = strValu$, $T^{\mathcal{L}_c/\theta_c}$ in e satisfies $\mathcal{L}_c/\theta_c = strValu$ and vice versa.*

The correctness of the lemma is guaranteed by the one-to-one correspondence between t's γ-trees and e's children.

Lemma 4. *Given a tuple $t =< x_t, x_c, \cdots >\in fortup(V)$ and its $\mathfrak{e}$-tree e, let T be a tree identified by $x_t/\gamma_t/\theta_t$ in t and T' be the corresponding tree identified by $\mathcal{L}_t/\theta_t$ in e. Obviously $T = T'$. As δS and δV have the same update action, if x_c satisfies the update condition, $\delta S(T) = \delta V(T')$.*

Theorem 1. *Update δS is a precise translation of the view update δV if (i) $\mathcal{L}_t/\theta_t \neq \phi$ and $x_c = x_t$, and (ii) $x_t/\gamma_t/\theta_t$ does not proceed any path in the where clause of δS.*

Proof. We follow Definition 6. Without losing generality, we assume that $x_t = x_c = x_1$. Figure 2 illustrates the relationship between a variable binding x_1 in the tuple $< x_1, \cdots >$ and the $\mathfrak{e}$-tree built from the tuple. The γ-trees T^{x_1/γ_t} and T^{x_1/γ_c} in x_1 become the children of e in the view. $T^{x_1/\gamma_t/\theta_t}$ and $T^{x_1/\gamma_c/\theta_c}$are an update target tree and a condition tree respectively. $T^{x_1/\gamma_t/\theta_t}$'s children will be deleted or a new child will be inserted.

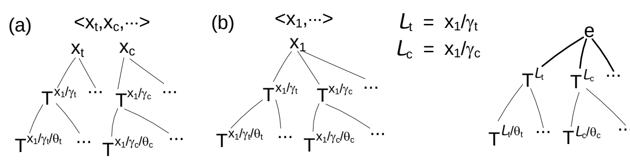

Figure 2: Each of tuples is mapped to an $\mathfrak{e}$-tree

(1) Correctness: $V(\delta S(S^i)) = \delta V(V(S^i))$

Consider two tuples $t_1 =< x_1^{(1)}, \cdots >$ and $t_2 =< x_1^{(2)}, \cdots >$ in the evaluation of δS where $x_1^{(1)}$ and $x_1^{(2)}$ are copies of x_1. Obviously if $x_1^{(1)}$ is updated, $x_1^{(2)}$ is updated too. That is, their source x_1 will be updated twice although only the first is effective. As δS and V have the same for clause, t_1 and t_2 exist in $fortup(V)$. Assume e_1 and e_2 are mapped from t_1 and t_2 respectively by V. Then, either both e_1 and e_2 are updated or none is updated.

$\supseteq$: Let $T^{\mathcal{L}_t/\theta_t}$ be a tree in an $\mathfrak{e}$-tree e of $V(S^i)$ updated to $\bar{T}^{\mathcal{L}_t/\theta_t}$ by δV (e becomes e' after the update). We show that $\bar{T}^{\mathcal{L}_t/\theta_t}$ is in e' of $V(\delta S(S^i))$. In fact, that $T^{\mathcal{L}_t/\theta_t}$ is in $V(S^i)$ means that there exists one and only one tuple $t = <x_1, \cdots>$ in $fortup(V)$ satisfying $cdn()$, that in the tuple, $x_1/\gamma_t/\theta$ identifies the source tree $T^{x_1/\gamma_t/\theta_t}$ of $T^{\mathcal{L}_t/\theta_t}$. $T^{\mathcal{L}_t/\theta_t}$ being updated by δV means that there exists a condition tree $T^{\mathcal{L}_c/\theta_c}$ in e and the condition tree satisfies $v/\mathfrak{e}/\mathcal{L}_c/\theta_c = strValu$.

On the other side, because V and δS have the same for clause, t is in $fortup(\delta S)$. Because $T^{\mathcal{L}_c/\theta_c}$ makes $v/\mathfrak{e}/\mathcal{L}_c/\theta_c = strValu$ true, so $T^{x_1/\gamma_c/\theta_c}$ makes $x_1/\gamma_c/\theta_c = strVal$ true (Lemma 3). This means $T^{x_1/\gamma_t/\theta_t}$ is updated by δS and becomes $\hat{T}^{x_1/\gamma_t/\theta_t}$. Thus t becomes $t' =< \bar{x}_1, \cdots >$. Because of Lemma 4, $\bar{T}^{x_1/\gamma_t/\theta_t} = \hat{T}^{x_1/\gamma_t/\theta_t}$. Because of (ii) of the theorem and Lemma 2, t' satisfies $cdn()$ and generalizes e' in the view. So $\bar{T}^{\mathcal{L}_t/\theta_t}$ is in $V(\delta S(S^i))$.

$\subseteq$: Let $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ be two trees in $V(\delta S(S^i))$ and their source tree(s) are updated by δS. We show that $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ are in $\delta V(V(S^i))$. There are three cases: (a) $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ share the same source tree $T^{x_1/\gamma_t/\theta_t}$ (they must appear in different $\mathfrak{e}$-trees in the view), and (b) $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ have different source trees $T_1^{x_1/\gamma_t/\theta_t}$ and $T_2^{x_1/\gamma_t/\theta_t}$. Case (b) has two sub cases: (b.1) $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ appear in the same $\mathfrak{e}$-tree in the view, and (b.2) $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ appear in different $\mathfrak{e}$-trees.

Case (a): That $T^{x_1/\gamma_t/\theta_t}$ is updated by δS means that there exist two tuples $<x_1^{(1)}, \cdots>$ and $<x_1^{(2)}, \cdots>$ in $fortup(\delta S)$ such that $x_1^{(1)} = x_1^{(2)}$, both tuples satisfy $cdn()$, and there exists condition tree $T^{x_1/\gamma_c/\theta_c}$ in each tuple satisfying $x_c/\gamma_c/\theta_c = strValu$, $T^{x_1/\gamma_t/\theta_t}$ is updated to $\bar{T}^{x_1/\gamma_t/\theta_t}$ by δS (two update attempts with the same action for the two tuples, only the effect of the first attempt is taken). After the update, the tuples become $t_1' = <\bar{x}_1^{(1)}, \cdots>$ and $t_2' = <\bar{x}_1^{(2)}, \cdots>$. By Lemma 2, t_1' and t_2' satisfy cdn of V and produce $e_1, e_2 \in V(\delta S(S^i))$ and $\bar{T}_1^{\mathcal{L}_t/\theta_t} \in e_1$ and $\bar{T}_2^{\mathcal{L}_t/\theta_t} \in e_2$.

On the other side, when V is evaluated against S^i, x_1 is copied to two tuples $t_1 = <x_1^{(1)}, \cdots>$ and $t_2 = <x_1^{(2)}, \cdots>$ in $fortup(V)$ and each of the tuples satisfies $cdn()$. They produce $\mathfrak{e}$-trees e_1' and e_2'. Because each tuple has a condition tree $T^{x_1/\gamma_c/\theta_c}$ satisfying $x_c/\gamma_c/\theta_c = strValu$, by Lemma 3, each of e_1' and e_2' has $T^{\mathcal{L}_c/\theta_c}$ satisfying $\mathcal{L}_c/\theta_c = strValu$ and each has a $T^{\mathcal{L}_t/\theta_t}$. Thus $T_1^{\mathcal{L}_t/\theta_t} \in e_1'$ and $T_2^{\mathcal{L}_t/\theta_t} \in e_2'$ will be updated to $\bar{T}_1^{\mathcal{L}_t/\theta_t}$ and $\bar{T}_2^{\mathcal{L}_t/\theta_t}$ by δV. e_1' and e_2' become e_1 and e_2 in $\delta V(V(S^i))$.

Case (b.1): That $T_1^{x_1/\gamma_t/\theta_t}$ and $T_2^{x_1/\gamma_t/\theta_t}$ are updated by δS and that they appear in different $\mathfrak{e}$-trees mean that there are two tuples $<x_{1[1]}, \cdots>$ and $<x_{1[2]}, \cdots>$ where $x_{1[1]}$ and $x_{1[2]}$ are different bindings of x_1, $T_1^{x_1/\gamma_t/\theta_t} \in x_{1[1]}$, $T_2^{x_1/\gamma_t/\theta_t} \in x_{c[2]}$, and each of tuples satisfies $cdn()$ and $x_c/\gamma_c/\theta_c = strValu$. $T_1^{x_1/\gamma_t/\theta_t}$ and $T_2^{x_1/\gamma_t/\theta_t}$ become $\bar{T}_1^{x_1/\gamma_t/\theta_t}$ and $\bar{T}_2^{x_1/\gamma_t/\theta_t}$ after the update and mapped to $\bar{T}_1^{\mathcal{L}_t/\theta_t}$ and $\bar{T}_2^{\mathcal{L}_t/\theta_t}$ in two different $\mathfrak{e}$-trees of $V(\delta S(S^i))$. Following the same argument of Case (a), $\bar{T}_1^{\mathcal{L}_t/\theta_t}$ and $\bar{T}_2^{\mathcal{L}_t/\theta_t}$ are in $\delta V(V(S^i))$.

Case (b.2): That $T_1^{x_1/\gamma_t/\theta_t}$ and $T_2^{x_1/\gamma_t/\theta_t}$ are updated by δS and that they appear in a single $\mathfrak{e}$-tree mean that there is one and only one tuple $<x_1, \cdots>$ where $T_1^{x_1/\gamma_t/\theta_t}, T_2^{x_1/\gamma_t/\theta_t} \in x_1$. The tuple satisfies $cdn()$ and there is a tree $T^{x_1/\gamma_c/\theta_c}$ in the tuple satisfying $x_1/\gamma_c/\theta_c = strValu$. $T_1^{x_1/\gamma_t/\theta_t}$ and $T_2^{x_1/\gamma_t/\theta_t}$ become $\bar{T}_1^{x_1/\gamma_t/\theta_t}$ and $\bar{T}_2^{x_1/\gamma_t/\theta_t}$ after the update and mapped to $\bar{T}_1^{\mathcal{L}_t/\theta_t}$ and $\bar{T}_2^{\mathcal{L}_t/\theta_t}$ in a single $\mathfrak{e}$-tree of $V(\delta S(S^i))$. On the other side, as $T_1^{x_1/\gamma_t/\theta_t}$ and $T_2^{x_1/\gamma_t/\theta_t}$ are mapped to a single $\mathfrak{e}$-tree e and share the same condition tree $T^{x_1/\gamma_c/\theta_c}$, $T_1^{\mathcal{L}_t/\theta_t}$ and $T_2^{\mathcal{L}_t/\theta_t}$ share the same condition tree $T^{\mathcal{L}_c/\theta_c}$ in e and will be updated by δV. So $\bar{T}_1^{\mathcal{L}_t/\theta_t}$ and $\bar{T}_2^{\mathcal{L}_t/\theta_t}$ are in the $\mathfrak{e}$-tree of $\delta V(V(S^i))$.

(2) δS is minimal

We prove by contrapositive. Let $T^{\mathcal{L}_t/\theta_t}$ be a tree in the view updated by δV. Then from above proofs, $T^{x_1/\gamma_t/\theta_t}$ is updated by δS and there exists a tuple $<x_1, \cdots>$ such that $T^{x_1/\gamma_t/\theta_t}$ is in x_1 and x_1 has a condition tree $T^{x_1/\gamma_c/\theta_c}$ satisfying "$cdn()$ and $x_1/\gamma_c/\theta_c = strValu$".

If $T^{x_1/\gamma_t/\theta_t}$ is not updated by $\delta S'$, either (a) x_1 is not a variable in the for-clause of $\delta S'$, i.e., x_1 is not in any tuple and neither is $T^{x_1/\gamma_t/\theta_t}$, or (b) x_1 is in the tuple $<x_1, \cdots>$ but $T^{x_1/\gamma_t/\theta_t}$ is not in x_1, or (c) x_1 is in the tuple $<x_1, \cdots>$ and $T^{x_1/\gamma_t/\theta_t}$ is in x_1 but one of "$cdn()$" and "$x_c/\gamma_c/\theta_c = strValu$" is not in $\delta S'$.

In Case (a), because x_1 is not a variable in $\delta S'$, so $T^{x_1/\gamma_t/\theta_t}$ will not be updated by $\delta S'$ (this does not prevent $T^{x_1/\gamma_t/\theta_t}$ from appearing in the view). This means that the $T^{\mathcal{L}_t/\theta_t}$ in $V(\delta S'(S^i))$ is different from the $T^{\mathcal{L}_t/\theta_t}$ in $\delta V(V(S^i))$

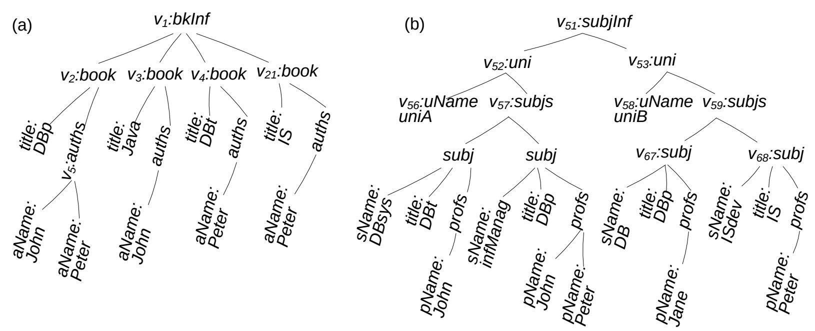

Figure 3: Books and their references

because the assumption assumes that the $T^{\mathcal{L}_t/\theta_t}$ in $\delta V(V(S^i))$ is updated. This contradicts the correctness of $\delta S'$.

In Case (b), because $T^{x_1/\gamma_t/\theta_t}$ is not in x_1, so $T^{x_1/\gamma_t/\theta_t}$ is not in $V(S^i)$. This contradicts the assumption that $T^{\mathcal{L}_t/\theta_t}$ is in the view.

In Case (c), if $cdn()$ is violated, the tuple of $T^{x_1/\gamma_t/\theta_t}$ will not be selected by V, so $T^{x_1/\gamma_t/\theta_t}$ is not in $V(S^i)$ which contradicts the assumption. If $x_1/\gamma_c/\theta_c = strValu$ is violated, $T^{x_1/\gamma_t/\theta_t}$ will not be updated by δV. This contradicts the assumption that $T^{\mathcal{L}_t/\theta_t}$ is updated by δV.

This concludes that δS is a precise translation.

□

We note that the theorem gives only a necessary condition but not a sufficient condition. The reason is that there exists other cases where a view update is translatable. These will be further presented in the following sections.

We use an example to show how a view update is translated using the results. Figure 3 shows two XML documents. Document (a) stores book information where *auths* and *aName* mean authors and author-name elements respectively. Document (b) stores university subject, textbook and professor information where *uName*, *subjs*, *sName*, *profs*, and *pName* mean university-name, subjects, subject-name, professors, and professor-name respectively.

The view *Qbk* is defined below to contain, for each use of a book by a university subject, the author names and the title of the book, the name of the university and the professors using the book in their teaching.

```
<Qbk>{  for x in doc("bkInf.xml")/bkInf/book,
            y in doc("subjInf.xml")/subjInf/uni,
            z in y/subjs/subj
        where x/title=z/title
        return <use>{x/auths}{x/title}{y/uName}{z/profs}</use>
}</Qbk>
```

The view instance for the XML documents is shown in Figure 4.

Now assume that the user of the view wants to add author *Susan* to the textbook *IS* in the view using the update statement below.

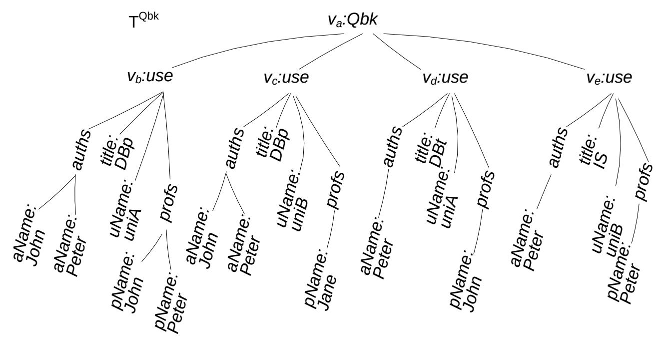

Figure 4: author-books and universities using them

```
for r in view(Qbk)/Qbk/use
where r/title="IS"
update r/auths { insert <aName>Susan</aName>}
```

With this statement, the user expects that next time when the view is selected, the output is Figure 5(a) where trees v_b, v_c and v_d are the same as those of Figure 4 and tree v_e contains the newly added author *Susan*.

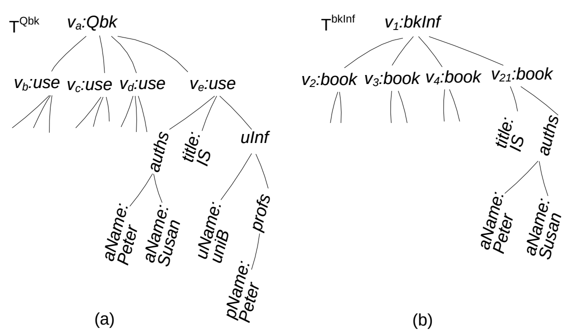

Figure 5: An insertion update

In the update statement, the update condition path and the update target path are $r/title$ and $r/auths$. The full view paths of the two paths are: $Qbk/use/title$ and $Qbk/use/auths$. In the paths, Qbk is v of Formula (2), use is $\mathfrak{e}$, $title$ is $\mathcal{L}_c$, $auths$ is $\mathcal{L}_t$, and θ_c and θ_t are ϕ. Following Procedure 1 by using $title$ and $auths$, we find the expressions $x/title$ and $x/auths$. By Algorithm 1, the following source update is derived:

```
for x in doc("bkInf.xml")/bkInf/book,
    y in doc("subjInf.xml")/subjInf/uni,
    z in $y/subjs/subj
where x/title=z/title and x/title="IS"
update x/auths { insert <aName>Susan</aName>}
```

When this statement is executed against Figure 3(a), the document becomes Figure 5(b) where trees v_2, v_3 and v_4 are the same as those in Figure 3(a) and v_{21} is changed. The view instance will appear as expected by the user when selected next time.

4 Update Translation when $\mathcal{L}_t/\theta_t \neq \phi$ and $x_c \neq x_t$

We look into the translation problem when the mappings of the update condition path and the update target path are led by different variables. The results of this section generalize the view update problem in the relational views when they are defined with the join operator.

In general, **view updates are not translatable in the case of $x_c \neq x_t$.** Consider two tuples where the binding x_t is copied to $x_t^{(1)}$ and $x_t^{(2)}$ to combine with two bindings $x_{c[1]}$ and $x_{c[2]}$ of x_c by the context-based production as

$$< \cdots, x_t^{(1)}, \cdots, x_{c[1]}, \cdots >$$
$$< \cdots, x_t^{(2)}, \cdots, x_{c[2]}, \cdots >$$

Assume that in the view, the update condition $x_c/\gamma_c/\theta_c$ is satisfied in $x_{c[1]}$ by violated in $x_{c[2]}$. Then, the copy of x_t corresponding to the first tuple will be updated but the one to the second tuple will not. In the source, if x_t is updated, not only the first copy of x_t changes, but also the second copy. In other words, the translated source update has view side-effect. However, if x_t in the source is not updated, all its copies in the view will not be changed.

Although generally view updates, when $x_c \neq x_t$, are not translatable, for the following view update, a precise translation exists.

$$\begin{array}{lll} V: & <v>\{ & \texttt{for } x_1 \texttt{ in } p_1, \quad \cdots, \quad x_n \texttt{ in } p_n \\ & & \texttt{where } \cdots \texttt{ and } x_c/\gamma_c/\theta_c = x_{c+1}/\gamma_{c+1}/\theta_{c+1} \texttt{ and } \cdots \\ & & \texttt{return } rtn(x_1, \cdots, x_n) \quad \}</v> \end{array} \tag{6}$$

where x_c/γ_c is in $rtn(x_1, \cdots, x_n)$, i.e., $x_c/\gamma_c/\theta_c$ is exposed in the view.

δV:

$$(\bar{p_s}, \quad v/\mathfrak{e}/\mathcal{L}_c/\theta_c = strValu, \quad v/\mathfrak{e}/\mathcal{L}_t/\theta_t, \quad del(T)|ins(T)) \tag{7}$$

where x_t is either x_c or x_{c+1}.

The condition requires that, in the view definition, x_c/γ_c must be a front part of one of the join path $x_c/\gamma_c/\theta_c$. At the same time, the path in view mapped from $x_c/\gamma_c/\theta_c$ must be the update condition path. Furthermore, the mapping of the update target path must be led by the same variable x_c leading the update condition path or by the variable x_{c+1} that joins x_c in the view definition.

Consider Example 1. With the condition, $y/D = z$ and $z =$ “1”, in the *where* clause, for a view update to be translatable, the mapping $x_c/\gamma_c/\theta_c$ of the view update condition path must be z or y/D, and the mapping $x_t/\gamma_t/\theta_t$ of the view update target path must be ended with F, G or E. We note that if $x_t/\gamma_t/\theta_t$ is ended with C or H, then $x_t/\gamma_t/\theta_t$ is a prefix of one of the paths in the join condition and the update will not be translatable.

Theorem 2. *Given the view V and a view update δV defined above, update δS of Formula (5) is a precise translation of the view update δV if (i) $\mathcal{L}_t/\theta_t \neq \phi$, and (ii) $x_c/\gamma_t/\theta_t$ does not proceed any path in the* where *clause of δS.*

Proof. The notation of this proof follows that of the proof for Theorem 1 and Figure 2. Consider two tuples $t_1 = <x_t^{(1)}, x_{c[1]}, \cdots>$ and $t_2 = <x_t^{(2)}, x_{c[2]}, \cdots>$ in the evaluation of $\delta S(S)$ where $x_t^{(1)}$ and $x_t^{(2)}$ are copies of x_t and $x_{c[1]}$ and $x_{c[2]}$ can be the same. If one is updated by δS, the other is updated too. The reason is that for $T_1^{x_t/\gamma_t/\theta_t} \in x_t^{(1)}$ and $T_2^{x_t/\gamma_t/\theta_t} \in x_t^{(2)}$, because of the join condition in Formula 6 $x_c/\gamma_c/\theta_c = x_{c+1}/\gamma_{c+1}/\theta_{c+1}$ and because $x_{c+1} = x_t$ and $x_t^{(1)} = x_t^{(2)}$, a condition tree $T_1^{x_c/\gamma_c/\theta_c}$ exists for $T_1^{x_t/\gamma_t/\theta_t}$ and $T_2^{x_c/\gamma_c/\theta_c}$ exists for $T_2^{x_t/\gamma_t/\theta_t}$ and $T_1^{x_c/\gamma_c/\theta_c} = T_2^{x_c/\gamma_c/\theta_c}$. Consequently if $T_1^{x_c/\gamma_c/\theta_c}$ satisfies the update condition, so does $T_2^{x_c/\gamma_c/\theta_c}$. So either both $T_1^{x_t/\gamma_t/\theta_t}$ and $T_2^{x_t/\gamma_t/\theta_t}$ are updated or none is updated. Following Lemma 4, if e_1 and e_2 are mapped from $T_1^{x_t/\gamma_t/\theta_t}$ and $T_2^{x_t/\gamma_t/\theta_t}$ respectively, if one is updated, the other is updated too.

The remaining proof can be completed by following the argument of the proof of Theorem 1. $\square$

5 Update Translation when $\mathcal{L}_t/\theta_t = \phi$

In this section, we identify translatable cases where $\mathcal{L}_t/\theta_t = \phi$, that is, the update target path is v or $v/\mathfrak{e}$. In the case of v, the update itself is an addition or a removal of an $\mathfrak{e}$-tree. In the case of $v/\mathfrak{e}$, the update is an insertion or a deletion of a γ-tree.

Obviously if the user does not know the structure of the view, wrong subtrees can be added. As an example, consider Q_1 in Figure 6. The path Q_1/E allows child elements labeled with C. If the user adds a sub tree labeled with F under v_u, the update violates the view definition. We exclude this type of cases and assume that the user knows the structure of the view and the updates aim to maintain such a structure.

In general, insertion updates are not translatable when $\mathcal{L}_t/\theta_t = \phi$. A number of reasons exist for this. The first is that there is no unique way to apply insertions to the source documents in many cases. The second reason is that the updates violate the context-based production. The third reason is there is no way for the user to write an update statement with a specific enough condition to update the view while the context-based production is not violated. We use three examples to illustrate the reasons.

Example 2. Consider Q_1 in Figure 6. If another subtree $(E\ (C\ (W:2)(G:8)))$ is inserted to Q_1, in the source the subtree $(C\ (W:2)(G:8))$ needs to be

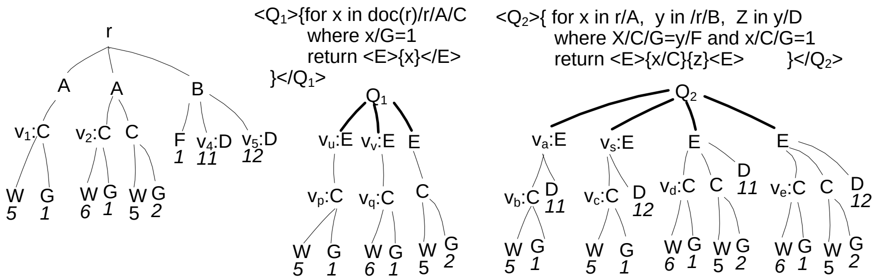

Figure 6: Two views to show updates to E and to Q

inserted to r. We cannot find a unique way to do so as the subtree can be inserted to an existing A element or a new A element is created and the subtree is inserted under the new A element.

Example 3. Consider Q_1 in Figure 6 again. If an update is an insertion of $(C\ (W:2)(G:8))$ under v_u, the context-based production is violated. By the context-based production, if x in the *return* clause is not followed by any path expression, only one C element is allowed in each E tree.

Example 4. Consider Q_2 in Figure 6 where C elements are selected by x/C in the *return* clause. If the user wants to insert another C element under both v_a and v_s (but not the other E elements) such that the context-based production is satisfied, the user has no way to specify an accurate condition for this because the node identifiers, v_a and v_s, are not available to the user.

For the same reasons, many deletion updates are not translatable. However in the case where all the expressions in the *return* clause start with the same variable, deletion updates to such views are translatable. We show the details below.

Let the view definition be

$$\begin{array}{l} V\texttt{:}\ \ <v>\{\ \texttt{for}\ x_1\ \texttt{in}\ p_1\texttt{,}\ \cdots\texttt{,}\ x_n\ \texttt{in}\ p_n \\ \qquad\quad \texttt{where}\ \ cdn(x_1,\cdots,x_n) \\ \qquad\quad \texttt{return}\ rtn(x_1)\quad \}</v> \end{array} \tag{8}$$

In the view, only the variable x_1 is involved in the *return* clause. Let the update statement to the view be

$$\begin{array}{l} \delta V\texttt{:}\ \texttt{for}\ e\ \texttt{in}\ v/\mathfrak{e} \\ \quad \texttt{where}\ e/\mathcal{L}_c/\theta_c = aVal \\ \quad \texttt{update}\ e\ \texttt{(delete}\ \mathcal{L}_t\texttt{)} \end{array} \tag{9}$$

The translated source update is

(10)

δS: for x_1 in p_1, $\cdots$, x_n in p_n
where $cdn(x_1, \cdots, x_n)$ and $x_1/\gamma_c/\theta_c = aVal$
update $x_1/\gamma_t/..$ (delete $\mathcal{L}_t$)

In the formulae, $\mathcal{L}_t$ is the last element of x_1/γ_t. To allow a $\mathcal{L}_t$ node to be inserted to or deleted from the source document, the target path must be $x_1/\gamma_t/..$.

Theorem 3. *Given the view definition V, the source update δS is a precise translation of the view update δV if $x_1/\gamma_t/..$ does not proceed any of the paths in the where clause of δS.*

Proof: We follow Definition 6 to prove $V(\delta S(S^i)) = \delta V(V(S^i))$ and omit the proof that δS is minimal. We note that $\mathcal{L}_t \neq \mathcal{L}_c$ implies $x_1/\gamma_c \neq x_1/\gamma_t$.

$\subseteq$: Assume that e'_1 and e'_2 are two $\mathfrak{e}$-trees in $V(\delta S(S^i))$. Then there exists two tuples $t'_1 =< \bar{x}_1^{(1)}, \cdots >$ and $t'_2 =< \bar{x}_1^{(2)}, \cdots >$ for e'_1 and e'_2 and they satisfy $cdn()$ of V. That the two tuples are updated by δS means that they are the results of updating two tuples $t_1 =< x_1^{(1)}, \cdots >$ and $t_2 =< x_1^{(2)}, \cdots >$ by $\delta S()$ and t_1 and t_2 satisfy $cdn()$ and have condition trees $T_1^{x_1/\gamma_c/\theta_c}$ and $T_2^{x_1/\gamma_c/\theta_c}$ satisfying $x_1/\gamma_c/\theta_c = aVal$, and the update deletes trees like T^{x_1/γ_t}. Consequently $T^{\mathcal{L}_t}$s are not in e'_1 and e'_2.

On the other side, as t_1 and t_2 satisfies $cdn()$, they produce e_1 and e_2 in $V(S^i)$. At the same time, e_1 and e_2 have condition trees $T_1^{\mathcal{L}_c/\theta_c}$ and $T_2^{\mathcal{L}_c/\theta_c}$ satisfying $\mathcal{L}_c/\theta_c = aVal$ (Lemma 3), they are updated as $T^{\mathcal{L}_t}$s will be deleted from from them. So they become e'_1 and e'_2 and are in $\delta V(V(S^i))$.

$\supseteq$: Let e'_1 and e'_2 be $\mathfrak{e}$-trees in $\delta V(V(S^i))$. Then there exist e_1 and e_2 in $V(S^i)$ and δV deletes $T^{\mathcal{L}_t}$s from them. That is, e_1 and e_2 have condition trees satisfying $cdn()$ and $\mathcal{L}_c/\theta_c = aVal$. e_1 and e_2 are for two tuples $t_1 =< x_1^{(1)}, \cdots >$ and $t_2 =< x_1^{(2)}, \cdots >$ in V and the two tuples satisfy $cdn()$.

On the other side, t_1 and t_2 satisfy $cdn()$ and $x_1/\gamma_c/\theta_c = aVal$ (Lemma 3), they will be updated and $T^{\mathcal{L}_t}$s will be deleted from them. So because of Lemma 2, they become $t'_1 =< \bar{x}_1^{(1)}, \cdots >$ and $t'_2 =< \bar{x}_1^{(2)}, \cdots >$. When $\delta S(S^i)$ is evaluated against V, t'_1 and t'_2 produces e'_1 and e'_2 which do not contain any $T^{\mathcal{L}_t}$s. So they are in $V(\delta S(S^i))$

δS is minimal: If a tree is not relevant to the view, the tree does not satisfy $cdn(x_1, \cdots, x_n)$ and it will not be updated by δS. $\square$

For the same view definition V in (5), if the update is applied to the root node as the following,

δV: for u in v,
where $u/\mathfrak{e}/\mathcal{L}_c/\theta_c = aVal$ (11)
update u (delete $\mathfrak{e}$)

the translated soruce update is

δS: for x_1 in p_1, $\cdots$, x_n in p_n
where $cdn(x_1, \cdots, x_n)$ and $x_1/\gamma_c/\theta_c = aVal$ (12)
update $x_1/..$ (delete $L(x_1)$)

We note that when an $\mathfrak{c}$ node is deleted, deleting all the γ trees from their parent nodes in the source document is not enough. The binding of the variable must be deleted.

Theorem 4. *Given view definition V in Formula (8), the source update δS in Formula (12) is a precise translation of the view update δV in Formula (11).*

proof: Let $t_1 =< x_1^{(1)}, \cdots >$ and $t_2 =< x_1^{(2)}, \cdots >$ be two tuples in $fortup(V)$, $x_1^{(1)}$ and $x_1^{(2)}$ be two copies of x_1 in the source, e_1 and e_2 be two $\mathfrak{c}$-trees for the tuples in $V(S^i)$, and e_1 and e_2 are deleted by δV. Because e_1 and e_2 are in $V(S^i)$, t_1 and t_2 satisfy $cdn()$. e_1 and e_2 being deleted by δV means that each of them has a subtree $T^{\mathcal{L}_c/\theta_c}$ satisfying $\mathcal{L}_c/\theta_c = aVal$. By Lemma 3, each of t_1 and t_2 has a tree $T^{x_1/\gamma_c/\theta_c}$ satisfying $x_1/\gamma_c/\theta_c = aVal$. Thus t_1 and t_2 will be updated by δS meaning the binding of x_1 will be deleted from the source. Consequently t_1 and t_2 will not be in $fortup(V(\delta S())$ and e_1 and e_2 will not be in $V(\delta S())$.

The proof that δS is minimal is similar to that of Theorem 1. $\square$

6 Conclusion

In this paper, we defined the view update problem in XML and shown the factors determining the translation problem. We identified the cases where view updates are translatable, shown a translation algorithm, gave the translated source updates, and proved the source updates are precise.

The translatability of view updates is information dependent. In this paper, we assume the only information available is the view definition and the update. When other information like keys and references are used in the translation, different algorithms and different source updates may be obtained. We leave the investigation of these problems as future work.

References

[1] F. Bancilhon and N. Spyratos. Update semantics of relational views. **T**ODS, 6(4):557–575, 1981.

[2] Scott Boag, Don Chamberlin, Mary F. Fernndez, Daniela Florescu, Jonathan Robie, and Jrme Simon. Xquery 1.0: An xml query language. **h**ttp://www.w3.org/TR/xquery/, 2007.

[3] Vanessa P. Braganholo, Susan B. Davidson, and Carlos A. Heuser. From xml view updates to relational view updates: old solutions to a new problem. **V**LDB Conference, pages 276–287, 2004.

[4] Gao Cong. Query and update through xml views. **L**NCS 4777 - DNIS 2007, page 8195, 2007.

[5] Gilles Falquet, Luka Nerima, and Seongbin Park. Hypertext view update problem. **T**echnical Report, University of Geneva, www.cui.unige.ch/isi/reports/hvu.ps, 2000.

[6] Peter Fankhauser. Xquery formal semantics state and challenges. **S**IGMOD Record, 30(3):14–19, 2001.

[7] Dongxi Liu, Zhenjiang Hu, and Masato Takeichi. Bidirectional interpretation of xquery. **P**EPM, pages 21–30, 2007.

[8] Yoshifumi Llasunaga. A relational database view update translation mechanism. **V**LDB Conference, pages 309–320, 1984.

[9] Igor Tatarinov, Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. Updating xml. **S**IGMOD conference, pages 413–424, 2001.

[10] Anthony Tomasic. View update translation via deduction and annotation. **I**CDT, pages 338–352, 1988.

[11] Anthony Tomasic. Determining correct view update translations via query containment. **W**orkshop on Deductive Databases and Logic Programming, pages 75–83, 1994.

[12] Ling Wang, Elke A. Rundensteiner, and Murali Mani. Updating xml views published over relational databases: towards the existence of a correct update mapping. **D**KE, 58:263–298, 2006.